



NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

DISSERTATION

**IDENTIFYING AND QUANTIFYING
EMERGENT BEHAVIOR THROUGH SYSTEM
OF SYSTEMS MODELING AND SIMULATION**

by

Mary Ann Cummings

September 2015

Dissertation Supervisor:

Man-Tak Shing

Approved for public release; distribution is unlimited

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			<i>Form Approved OMB No. 0704-0188</i>	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington, DC 20503.				
1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE September 2015	3. REPORT TYPE AND DATES COVERED Ph.D. Dissertation		
4. TITLE AND SUBTITLE IDENTIFYING AND QUANTIFYING EMERGENT BEHAVIOR THROUGH SYSTEM OF SYSTEMS MODELING AND SIMULATION			5. FUNDING NUMBERS	
6. AUTHOR(S) Cummings, Mary Ann				
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000			8. PERFORMING ORGANIZATION REPORT NUMBER	
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A			10. SPONSORING / MONITORING AGENCY REPORT NUMBER	
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government. IRB Protocol number ____N/A____.				
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited			12b. DISTRIBUTION CODE	
13. ABSTRACT (maximum 200 words) The U.S. Department of Defense builds many weapons systems as Systems of Systems (SoSs). Operational testing is used to test capabilities in these SoSs, but testing all configurations of the SoSs may not be possible due to cost and the environment. Modeling and simulation (M&S) is an alternative to operationally testing all functionality and interfaces in these SoSs. An inherent deficiency of existing M&S approaches, however, lies in the emergent behavior that occurs as a result of interactions among the component systems. Applying M&S to individual components cannot detect this emergent behavior, but not detecting emergent behavior of an SoS as a whole diminishes its testing process. This dissertation develops a software architecture for an M&S framework, based on Ziegler's theory, that makes the identification and quantification of emergent behavior possible in an SoS M&S by providing a collector of SoS metrics. This software architecture has a great advantage over the current approaches because it supports swappable and configurable components to make simulation possible over different models and parameters without costly reprogramming. This dissertation also develops an improved process for the analyst to identify and quantify emergent behavior in an SoS M&S.				
14. SUBJECT TERMS system of systems, modeling and simulation, simulators, experimental frames, emergent behavior, metrics collector, emergent behavior, SoS metrics			15. NUMBER OF PAGES 165	
			16. PRICE CODE	
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UU	

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

**IDENTIFYING AND QUANTIFYING EMERGENT BEHAVIOR THROUGH
SYSTEM OF SYSTEMS MODELING AND SIMULATION**

Mary Ann Cummings
Civilian, Department of the Navy
B.S., James Madison University, 1984
M.S., Naval Postgraduate School, 1990

Submitted in partial fulfillment of the
requirements for the degree of

DOCTOR OF PHILOSOPHY IN SOFTWARE ENGINEERING

from the

**NAVAL POSTGRADUATE SCHOOL
September 2015**

Approved by: Man-Tak Shing, Ph.D.
Associate Professor of
Computer Science,
Dissertation Supervisor

Thomas Otani, Ph.D.
Associate Professor of
Computer Science

James Bret Michael, Ph.D.
Professor of Computer Science

James Scrofani, Ph.D.
Associate Professor of
Computer Engineering

Preetam Ghosh, Ph.D.
Associate Professor of
Computer Science, Virginia
Commonwealth University

Mark Anderson, Ph.D.
Senior Engineer
Naval Surface Warfare Center,
Dahlgren Division

Approved by: Peter Denning, Ph.D., Chair, Department of Computer Science

Approved by: Douglas Moses, Ph.D., Vice Provost for Academic Affairs

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

The U.S. Department of Defense builds many weapons systems as Systems of Systems (SoSs). Operational testing is used to test capabilities in these SoSs, but testing all configurations of the SoSs may not be possible due to cost and the environment. Modeling and simulation (M&S) is an alternative to operationally testing all functionality and interfaces in these SoSs. An inherent deficiency of existing M&S approaches, however, lies in the emergent behavior that occurs as a result of interactions among the component systems. Applying M&S to individual components cannot detect this emergent behavior, but not detecting emergent behavior of an SoS as a whole diminishes its testing process. This dissertation develops a software architecture for an M&S framework, based on Ziegler's theory, that makes the identification and quantification of emergent behavior possible in an SoS M&S by providing a collector of SoS metrics. This software architecture has a great advantage over the current approaches because it supports swappable and configurable components to make simulation possible over different models and parameters without costly reprogramming. This dissertation also develops an improved process for the analyst to identify and quantify emergent behavior in an SoS M&S.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

I.	INTRODUCTION.....	1
A.	MOTIVATION	6
B.	PROBLEM STATEMENT	7
C.	BASIS FOR PROPOSED SOLUTION.....	8
D.	PROPOSED SOLUTION.....	11
E.	HYPOTHESIS.....	15
F.	SPECIFIC GOALS OF PROPOSED RESEARCH	15
G.	NEW CONTRIBUTIONS.....	15
H.	RESEARCH STRATEGY AND METHODS	16
	1. Research Questions.....	16
	2. Research Approach and Methodology.....	16
I.	DISSERTATION ORGANIZATION.....	17
 II.	 BACKGROUND	 19
A.	SOFTWARE ARCHITECTURES.....	19
	1. SoS Architectures.....	19
	2. Components and Connectors Architectural Style	20
	3. Publish/Subscribe Architectural Style	22
B.	MODELING AND SIMULATION THEORY BY ZEIGLER.....	22
	1. Experimental Frame.....	23
	2. Simulator	24
	a. <i>DEVS</i>	24
	b. <i>DTSS</i>	25
	c. <i>DESS</i>	25
	3. Model.....	26
C.	REUSABILITY OF DEVS COMPONENTS	26
D.	MODELING TYPES.....	27
E.	AGENT-BASED MODELING AND SIMULATION	28
F.	EMERGENT BEHAVIOR	30
G.	INTERACTION METRICS	32
H.	M&S INTEROPERABILITY	33
I.	MODELING AND SIMULATION SOFTWARE	
	FRAMEWORKS AND LANGUAGES	33
	1. Frameworks.....	33
	a. <i>Orchestrated Simulation through Modeling (OSM)</i>	33
	b. <i>NetLogo</i>	35
	c. <i>Simulink through MATLAB</i>	37

d.	<i>Systems Tool Kit</i>	38
e.	<i>Ptolemy</i>	39
2.	Modeling Languages	40
a.	<i>Systems Modeling Language (SysML)</i>	40
b.	<i>High Level Architecture (HLA)</i>	42
J.	SUMMARY	43
III.	METHODOLOGY	45
A.	AGENTS	48
B.	SWAPPABLE EXPERIMENTAL FRAMES	55
C.	SWAPPABLE SIMULATORS	63
D.	METRICS COLLECTOR	71
IV.	EMERGENT BEHAVIOR IDENTIFICATION AND QUANTIFICATION	79
A.	ANALYSIS NEEDS IDENTIFIED	80
B.	EXPERIMENT OBJECTIVES DETERMINED	80
C.	SCENARIOS DEFINED, EXPERIMENTAL FRAME AND SIMULATOR CHOSEN	81
D.	EXPERIMENT SETUP AND RUN	81
E.	EMERGENT BEHAVIOR IDENTIFIED AND QUANTIFIED	84
V.	PROOF OF CONTRIBUTIONS	87
A.	PREDATOR PREY SIMULATION	87
1.	Description of NetLogo Matched Version	89
2.	Comparison of NetLogo and OSM	97
3.	Simulators Used in the Closest Food Version of the Predator Prey Simulation	98
a.	<i>Discrete Event Version Using the Discrete Simulator</i>	98
b.	<i>Discrete Time Version Using the Discrete Simulator</i>	100
4.	Addition of a Weed Agent	103
5.	Simulation Driven by DEVS Simulator	105
6.	Simulation Driven by DTSS Simulator	107
7.	Simulation Driven by Discrete Simulator and Using the Lattice Experimental Frame	110
8.	Simulation Driven by Discrete Simulator and Using the Stochastic Experimental Frame	113
B.	EMERGENT BEHAVIOR IN PREDATOR PREY SIMULATION	115
C.	EXPERIMENTAL FRAMES	120

D.	PROOF OF HYPOTHESIS WITH PREDATOR PREY SIMULATIONS	121
E.	BMDS SIMULATION.....	123
F.	EVALUATION	125
G.	SUMMARY	126
VI.	CONCLUSION	129
A.	SUMMARY OF SIGNIFICANT FINDINGS	129
B.	FUTURE RESEARCH.....	133
	APPENDIX. PREDATOR PREY SIMULATION PSEUDO CODE.....	135
	LIST OF REFERENCES.....	139
	INITIAL DISTRIBUTION LIST	143

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF FIGURES

Figure 1.	Elements of the Ballistic Missile Defense System (BMDS)	2
Figure 2.	Modeling and Simulation Process	4
Figure 3.	Circular Role of Simulation in System Development.....	5
Figure 4.	Example of Multi-agency Weapon SoS Simulation Architecture	6
Figure 5.	M&S Theoretical Framework Entities and Their Relationships.....	10
Figure 6.	Modified M&S Theoretical Framework Entities and Their Relationships.....	14
Figure 7.	Example of C2 Style	21
Figure 8.	Adding Context to the Models to Allow Model Reusability	27
Figure 9.	Agent Elements	29
Figure 10.	OSM Software Architecture	35
Figure 11.	Movement of Agent in NetLogo's Predator Prey Example.....	37
Figure 12.	SysML Language Diagrams	41
Figure 13.	OSM Software Architecture with the Metrics Collector, Experimental frame Exchange and the Simulator Exchange.....	45
Figure 14.	Simulator Exchange/Experimental Frame Exchange/Metrics Collector Class Diagram	47
Figure 15.	Simulator Exchange/Experimental Frame Exchange/Metrics Collector Collaboration Diagram.....	48
Figure 16.	Predator Prey Example Class Diagram.....	51
Figure 17.	Predator Prey Example Class Diagram with Addition of Tractor Agent.....	52
Figure 18.	Agent Class Diagram	54
Figure 19.	Agent Class Diagram	54
Figure 20.	Example of a Multi Run Experimental Frame Input Panel.....	55
Figure 21.	Example of a Lattice Experimental Frame Input Panel	56
Figure 22.	Example of a Stochastic Experimental Frame Input Panel.....	56
Figure 23.	Experimental frame Exchange and Experimental frames Connected Via the C2 Architectural Style.....	59
Figure 24.	Example Input Parameter Panel.....	60
Figure 25.	Experimental Frame Exchange Class Diagrams.....	61

Figure 26.	Experimental Frame Exchange Use Case Description	62
Figure 27.	Experimental Frame Exchange Sequence Diagram.....	63
Figure 28.	Finite State Automaton	65
Figure 29.	Finite State-Time Automaton	66
Figure 30.	Swappable Simulator Architectural Style.....	67
Figure 31.	Simulator Exchange Class Diagrams.....	69
Figure 32.	Simulator Exchange Use Case Diagram.....	70
Figure 33.	Simulator Exchange Sequence Diagram.....	70
Figure 34.	Predator Prey Agent Metrics.....	72
Figure 35.	Predator Prey Interaction Metrics	73
Figure 36.	Metrics Collector in the Publish/Subscribe Architectural Style through OSM's Bulletin Board.....	75
Figure 37.	Metrics Collector Class Diagrams	76
Figure 38.	Metrics Collection Use Case.....	77
Figure 39.	Metrics Collector Sequence Diagram	77
Figure 40.	Emergent Behavior Identification and Quantification Process.....	80
Figure 41.	Agent Selection Example.....	82
Figure 42.	Another Agent Selection Example.....	82
Figure 43.	Example Input Panel	83
Figure 44.	Example Metrics to be Collected.....	83
Figure 45.	Run Button Location.....	84
Figure 46.	Results of One Run of a Simulation	85
Figure 47.	Results of a Stochastic 3 Run Graph.....	86
Figure 48.	The Elements of This Research For this Proof of Concept	89
Figure 49.	Start of Predator Prey simulation.....	90
Figure 50.	Rabbit FSA.....	91
Figure 51.	Wolf FSA	92
Figure 52.	Grass FSA	92
Figure 53.	Input Parameters for the Grass.....	93
Figure 54.	Input Parameters for the Grass Agent.....	94
Figure 55.	Input Parameters for the Rabbit	94
Figure 56.	Input Parameters for the Wolf Agent.....	94

Figure 57.	NetLogo Predator Prey Simulation.....	95
Figure 58.	Run 1 using OSM Framework with Random Movements to Match NetLogo Run.....	96
Figure 59.	Run 2 using OSM Framework with Random Movements to Match NetLogo Run.....	96
Figure 60.	Run 3 using OSM Framework with Random Movements to Match NetLogo Run.....	97
Figure 61.	Wolf and Rabbit FSA.....	99
Figure 62.	Simulation Based on FSA.....	100
Figure 63.	Wolf and Rabbit Discrete Time FS-TA.....	101
Figure 64.	Wolf and Rabbit FS-TA.....	101
Figure 65.	Grass FS-TA.....	102
Figure 66.	Grass FS-TA.....	102
Figure 67.	Simulation Based on FS-TAs.....	103
Figure 68.	Weeds Input Parameters Panel.....	103
Figure 69.	Results of Simulation with Addition of Weeds Agent.....	104
Figure 70.	Results of Simulation with Changes to Weeds Agent Parameters.....	105
Figure 71.	DEVS Simulation at Startup.....	106
Figure 72.	Graph of Results of DEVS Simulation.....	106
Figure 73.	Visualization at end of DEVS Simulation.....	107
Figure 74.	DTSS Simulation at Startup.....	108
Figure 75.	Graph of Results of the DTSS Simulation.....	109
Figure 76.	Visualization at end of DTSS Simulation.....	109
Figure 77.	Lattice Experimental frame Input Panels.....	111
Figure 78.	Graph of Each Agent Comparing Each Run.....	112
Figure 79.	Graph of All Agents for All Runs.....	112
Figure 80.	Stochastic Experimental frame Input Panel.....	113
Figure 81.	Graph of Each Run.....	114
Figure 82.	Graph Showing Results of All Four Stochastic Runs.....	114
Figure 83.	Graph of Metrics for DTSS Run.....	115
Figure 84.	Graph of Metrics for DEVS Run.....	116
Figure 85.	Visualization of DTSS Run at 10 Seconds.....	117
Figure 86.	Visualization of DTSS Run at 37 Seconds.....	117

Figure 87.	Visualization of DTSS Run at 60 Seconds	118
Figure 88.	Visualization of DEVS Run at 22 Seconds.....	118
Figure 89.	Visualization of DEVS Run at 45 Seconds.....	119
Figure 90.	Visualization of DEVS Run at 85 Seconds.....	119
Figure 91.	Graph of the Predator Prey Simulation Using a Four Run Lattice Experiment.....	120
Figure 92.	Graph of the Predator Prey Simulation Using a Three Run Stochastic Experiment.....	121
Figure 93.	BMDS Communications Simulation using the Discrete Simulator and the Multi Run Experimental frame	124
Figure 94.	Architectural Elements Used in Both Simulations	125

LIST OF TABLES

Table 1.	Differences in Simulation Methods	23
Table 2.	Types of Simulations Developed and Run.....	88

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF ACRONYMS AND ABBREVIATIONS

ABM	Agent Based Modeling
ACS	Aegis Combat System
BMDA	Ballistic Missile Defense-Agents
BMDS	Ballistic Missile Defense System
C2	Components and Connectors
C2BMC	Command, Control, Battle Management and Communications
DESS	Differential Equation System Specification
DEVS	Discrete Event System Specification
DIS	Distributed Interactive Simulation
DOD	Department of Defense
DTSS	Discrete Time System Specification
EBM	Equation Based Modeling
FSA	Finite State Automaton
FS-TA	Finite State-Time Automaton
GUI	Graphical User Interface
HLA	High Level Architecture
M&S	Modeling and Simulation
MATLAB	MATrix LABoratory
MBSE	Modeling Based Systems Engineering
NSWCDD	Naval Surface Warfare Center, Dahlgren Division
OSM	Orchestrated Simulation through Modeling
SoS	System of Systems
STK	Systems Tool Kit
SysML	Systems Modeling Language
TENA	Test and training Enabling Architecture
THAAD	Terminal High Altitude Area Defense
UAV	Unmanned Aerial Vehicle

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

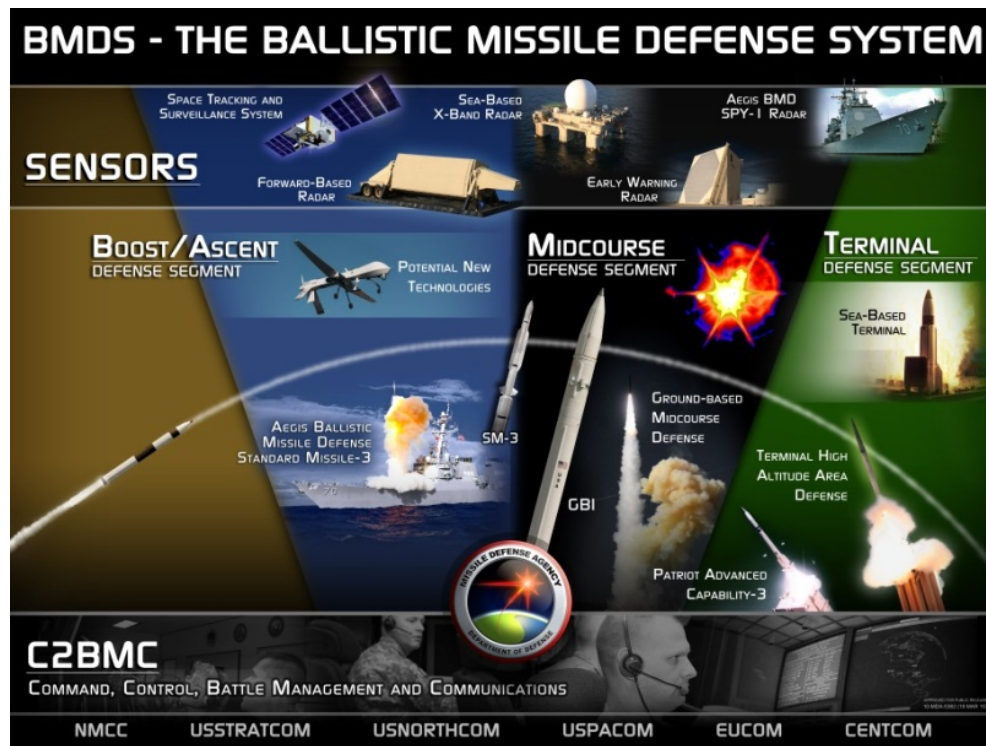
I wish to thank my dissertation committee for all their help and guidance through this process of achieving a Ph.D. in software engineering. Without them, this would not have been possible. My committee included Doctors Man-Tak Shing, Thomas Otani, Bret Michael, Jim Scrofani, Preetam Ghosh, and Mark Anderson. I want to call out Doctor Shing as my advisor and committee chair for all of his advice and coaching through this long work. I also want to thank Clint Winfrey of NSWCD for all his help in coding the resultant work in this dissertation. Clint, I could not have done this without you. I want to thank my management at NSWCD, Kim Payne, Patti Fetter, Everett Wiles, Dave Richardson, and Jim Wolfe, for all their support through this time. Lastly, I want to thank my family, John, Missy, Emily, and Mary Jo, for their patience and support through all the hours I spent working on this when I got home each night.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

We live in a complicated world—a place where simple systems are no longer the norm. Our weapon systems and the warfighter capabilities they provide are also complicated. Today’s weapon systems are complex, and we have observed in the United States that “an increasing number of warfighting capabilities are being achieved through a System of Systems (SoS) approach” (Director, Systems and Software Engineering and Deputy Under Secretary of Defense (Acquisition and Technology), 2008). An SoS can be described as a set of systems that bring their capabilities together to create a new system that provides more functionality than the sum of the component systems. SoSs may be formed by incorporating formerly standalone legacy weapon systems (with their own purposes, and for which it would be impractical budgetarily or otherwise to throw out and start afresh) with new developments to realize capabilities none of the individual systems can provide on their own. For example, the U.S. Navy’s Aegis Combat System (ACS) and the U.S. Army’s Terminal High Altitude Area Defense (THAAD) have been integrated with the newly developed Command, Control, Battle Management and Communications (C2BMC) system to become part of an SoS known as the Ballistic Missile Defense System (BMDS), shown in Figure 1. The mission of the BMDS is “an integrated, layered defense system to defend the United States, its deployed forces, and allies against all ranges of enemy ballistic missiles in all phases of flight” (The Ballistic Missile Defense System, 2015).

Figure 1. Elements of the Ballistic Missile Defense System (BMDS)



From *The Ballistic Missile Defense System (BMDS)*. (2015, January 15). Retrieved March 13, 2015, from U.S. Department of Defense Missile Defense Agency: <http://www.mda.mil/system/system.html>

In order to develop these complicated systems, the Policy for Systems Engineering in the Department of Defense (DOD), issued by the Under Secretary of Defense for Acquisition, Technology & Logistics in 2004, calls for “the application of a rigorous systems engineering discipline that meets the challenge of developing and maintaining needed warfighting capability” through integration of complex systems in SoSs. This discipline calls for the use of capabilities-based acquisition, which has become the basis for defining and analyzing the warfighter’s (also known as the user) needs. A key goal of engineering an SoS is to specify the value-added capabilities and ensure that these capabilities are correctly implemented. This may force the previously stand-alone systems to provide functionality or interfaces that had not been considered in their individual designs (Garrett, Anderson, Baron, & Moreland, 2010). In order to validate an SoS, the complex behaviors resulting from the interactions between the component systems must be known, tested and accepted.

Modeling and simulation (M&S) can be an affordable alternative to operationally testing all functionality and interfaces in a weapon SoS. Schaeffer's 2006 work, the Department of Defense Acquisition Modeling and Simulation Master Plan, calls for the use of M&S in defining, developing, testing, producing and sustaining warfighting capability that supports the spectrum of DOD missions. The use of M&S can be thought of as the virtual test for SoSs, thus reducing the need for operationally testing these weapons in all possible configurations. With changes in a few parameters, M&S can be used to test many configurations (many of which are cost prohibitive or just impossible to test via operational tests). This virtual testing is not a replacement for operationally testing, but it reduces the number of tests needed. If done properly, operationally testing provides data to the M&S, and the M&S shapes the tests needed to make the most of the money spent on these tests.

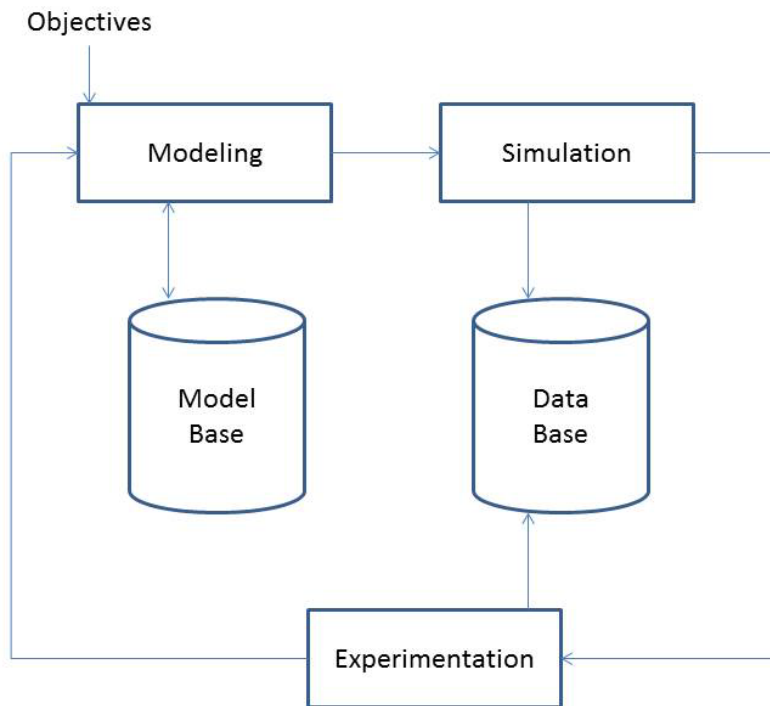
Zeigler (1984) calls for a never-ending M&S process (as shown in Figure 2) that ties the models, simulation and real system (under test) together. The process is driven by objectives of the system under test being generated from outside the system boundaries. These objectives do not happen just once, but appear and change over time. As new objectives come in, the modeling activity is started again. Zeigler tells us that the available knowledge for building the model comes from the model base. Model construction is followed by the simulation of the model. The data used in simulation comes from the database which stores and organizes data gathered about the real system. Zeigler says that the simulation then leads to new experimentation on the real system (otherwise known as operationally testing). As the process is traversed, refined objectives may be formulated as deficiencies in the current knowledge base become more apparent.

To do this never-ending process successfully, we must understand the behaviors that emerge from the interactions of the models. This behavior occurs as a result of interactions of its components (or parts), and is not found in any of the components alone. Chan (2011) defines this as a behavior or pattern that emerges or occurs as a result of interactions of its components (or parts). A defining characteristic is that the behavior does not appear within one component alone; it only appears in the interactions. This allows the SoS to be greater than the sum of its components. Talley (2008) claims that

this behavior provides uncertainty in the design of an SoS. Examples of emergent behavior are:

- The working of the human mind, which emerges from neurons collaborating together
- The flocking of birds
- Schools of fish

Figure 2. Modeling and Simulation Process



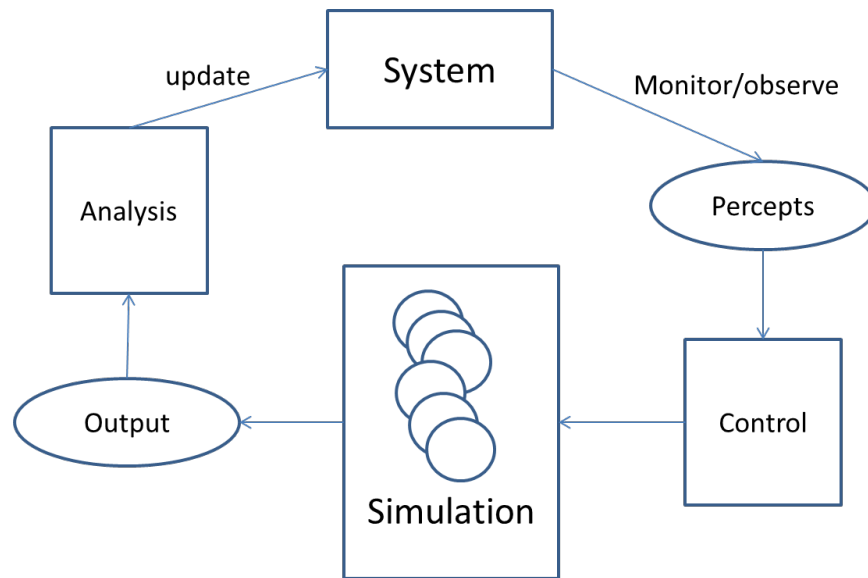
From Zeigler, B. (1984). Multifaceted Modeling and discrete Event Simulation. London: Academic Press.

The changing of the objectives may change the behavior that emerges from the M&S and from operationally testing. Changes that are not under the control of the M&S and operational test also can cause changes in the emergent behavior. An example of this type of change can be found in the environment (such as weather changes). These types of changes may be impossible to test in an operational test because they cannot be controlled. They can be tested in the M&S, however, because the tester has control of the experiment.

Yilmaz and Oren (2007) defines the term “modeling” in M&S as the act of developing a static representation of some real world object. A model can also be thought of as a well-defined representation of some real world object. The term “simulation” is “the execution of models over time representing the attributes of one or more entities or processes” (Yilmaz and Oren, 2009). According to Yilmaz and Oren (2007), “the purpose of M&S in dealing with complexity is not to predict the outcome of a system necessarily, but rather it is to reveal and understand the complex and aggregate system behaviors that emerge from the interactions of elements involved.” Most often, simulation is performed by a computer. Thus, simulation is facilitated by advances in system engineering and software agents, among other things.

Figure 3 shows that the simulation plays a vital (circular) role in the development of a system. According to Yilmaz (2004), the simulation is driven by data taken from the real system under control.

Figure 3. Circular Role of Simulation in System Development

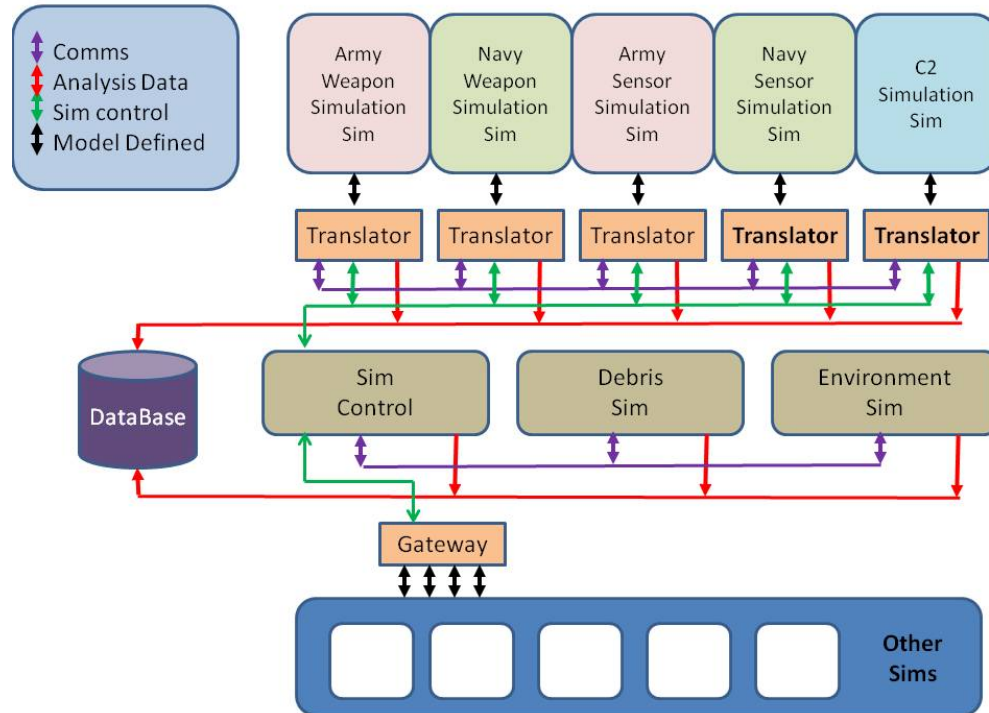


From Yilmaz, L. (2004). “On the need for contextualized introspective models to improve reuse and composability of defense simulations.” . *The Journal of Defense Modeling and Simulation: Applications, Methodology, Technology* 1.3 .

A. MOTIVATION

Many Department of Defense (DOD) organizations that develop weapon SoSs have a large set of M&S systems across various agencies that own the component weapon systems. These M&S systems represent the different component weapon systems, environments, threats, communications, debris, etc. They tie these pieces together with a framework built for that one particular simulation. See Figure 4 for an example of this type of simulation developed for a multi-agency weapon SoS simulation (such as a BMDS simulation).

Figure 4. Example of Multi-agency Weapon SoS Simulation Architecture



These component M&S systems were not originally developed to interface with other systems. They were built to run independently and each has its own event drivers and timing engines. When brought together into a SoS simulation, there can be multiple M&S systems running with different time clocks and time steps. There are also many event drivers passing events only to the one M&S component system, instead of having a centralized event driver.

These component M&S systems have their own scenarios (objectives of the M&S) built for them. There is no SoS scenario. The execution of the SoS simulation, then, is the creation of each of the scenarios for the component M&S systems, and running the simulations. This does not allow for collecting SoS metrics, which are necessary for the identification and quantification of emergent behavior of the target SoS.

The problems with this type of simulation are:

- Every time any of the component M&S systems change, the framework and any component M&S system that interfaces with that component also have to change. This is due to the tight coupling among components.
- Any new component M&S system will cause the other component M&S systems and the framework to change.
- The inputs and outputs may be different for every component M&S system because there are no standard interfaces between the component M&S systems. This leads to the framework changing every time any of the component M&S systems change inputs or outputs.
- Any scenario changes could cause changes to other component M&S systems and the framework.

This type of architecture does not allow for the identification of emergent behavior because:

- The component simulations are not linked together via standard interfaces.
- Swappability and reusability of components within and among simulations are lacking. This leads to tight coupling and changes become necessary to other components when a component changes.
- Each component simulation does not provide data to any type of System of System metrics.
- If any of the component simulations change, then other component simulations and the framework may have to change.

B. PROBLEM STATEMENT

M&S can be used for the virtual testing of SoS capabilities prior to development of a weapon SoS and after development is completed. To perform this virtual testing, we must be able to identify and quantify the behavior that emerges from the interactions among the component weapon systems. To do this, we need the ability to experiment

with different input values, different experimental frames, and different simulators to simulate more fully the weapon SoS complex environment, without changing the fundamental model components of the simulation. Zeigler (1976) tells us that an experimental frame “specifies the conditions under which the system to be modeled is to be observed.” The simulator drives the model through time. We also need the ability to experiment with different combinations of model components without changing the experimental frame and simulator.

C. BASIS FOR PROPOSED SOLUTION

Is there an M&S methodology that can allow for the identification and analysis of emergent behavior by allowing for standard interfaces, a separate event driver and timing driver, and a method for enabling the changing of one element of the simulation without changing any other elements? The book, *Theory of Modeling and Simulation*, originally published in 1976 by Dr. Zeigler, describes a theory that has brought coherence and unity to M&S. This was the first work to describe approaches to M&S as system specification formalisms. Although it has been over 35 years since this book was distributed (and two update editions have come out since then), this work on M&S formalisms and frameworks (a theoretical framework, not software framework) is still heavily influencing M&S today. The formalisms described in this work are:

- “Discrete Event System Specification (DEVS)”
- “Discrete Time System Specification (DTSS)”
- “Differential Equation System Specification (DESS), also known as continuous time”

In Zeigler (1984), the author provides the definition of a formalism as a convention in communication that specifies a class of objects under discussion in an unambiguous and general manner. It allows us to focus on features of objects which are relevant to our discussion. This process is called abstraction. Formalisms can be hard to grasp because the abstraction takes us out of the concrete world.

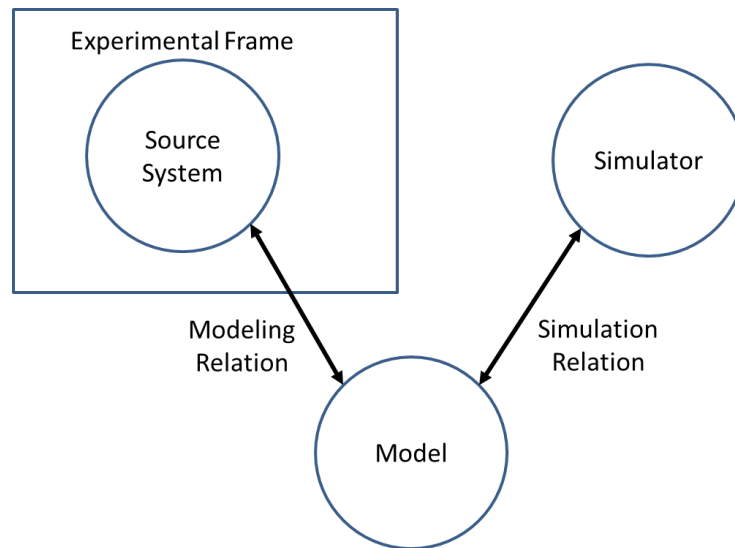
Zeiger (1976) defines a formalism as “containing a list of parameters which are a set of theoretic constructs and a list of constraints.” If an object is considered a part of a

formalism then this object has a set of parameters which can take any of the values that satisfy the constraints. The structure of the formalism refers to its parameters and constraints. The DEVS, DESS, and DTSS formalisms are described in section B of Chapter II.

Zeigler's work in 1984 describes the basic objects of the M&S theoretical framework as the source (or real) system, the model, the Simulator and the Experimental frame. The source system provides the data we want to observe (otherwise known as the system we want to model). To observe it, we must determine the conditions we want to observe in the real system. These conditions are specified in the Experimental frame, which documents the objectives of the model we want to build. A model is some representation of the real system. The model has the ability to execute instructions and generate behavior. The Simulator triggers these instruction executions and behavior through time. Theoretically, it is this "through time" capability that allows us to identify the emergent behavior via changes in SoS metrics as time changes. Figure 5 shows the basic M&S framework objects and their relationships as proposed in Zeigler (2000).

Zeigler (2000 & 2013) states that there is one Simulator per model (or class of models). This is one of the elements that makes this M&S theory stand out because it pulls the event driver and timing out of the models. This helps the current M&S situation described above, because it allows the overall SoS M&S engineer to build the Simulators while understanding the overall purpose/function of the SoS M&S. It also removes the need to build event drivers and timing into each separate component M&S system.

Figure 5. M&S Theoretical Framework Entities and Their Relationships



From Zeigler, B. P. (1976). *Theory of Modeling and Simulation*. New York. John Wiley.

Although this is a great improvement, this Simulator is tied to the model (or class of models). This means that there are unique Simulators for each model or class of models. For a weapon SoS M&S, this can mean a Simulator for every element in the weapon SoS (which could include weapon models, environment models, and threat models) because one can expect each component M&S system to be a different class of models. Taken further, this also means that the model may have to change when the Simulator changes. This does not help our emergent behavior problem because we still cannot standardize and gather SoS metrics in one place.

Zeigler (2000) states that “the Experimental frame is the operational formulation of the objectives” (resulting in the scenario) that drive a particular M&S. This helps the problem described above by allowing the development of one Experimental frame (holder of the scenario), built by the SoS M&S engineer, to be used by all of the component M&S systems instead of having a separate scenario for each M&S.

This means that there is a unique Experimental frame for each M&S. Also, there is a unique Experimental frame for each objective or purpose of a single M&S. In a weapon SoS, this could mean a unique Experimental frame for testing the range of the weapons, for measuring the performance of the weapons against a threat, and for

analyzing environment changes on the weapons. According to Zeigler (2000), “the Experimental frame is the formulation of the objectives that motivate the M&S project.” This indicates that the model and Simulator may change if the objectives change. This does not help our emergent behavior problem because we need to initialize our SoS metrics in one place and make this initialization reusable in order to create a standardized method of identifying and analyzing emergent behavior. If we have unique Experimental frames for each objective, then we have to initialize our SoS metrics in each frame. This does not allow for reuse.

Taken together, for one System of Systems M&S (and this is true for our DOD example above), we could have many Simulators and many Experimental frames. For example, for a BMDS simulation with twelve weapon and satellite models, we could have twelve Simulators and fifty or more Experimental frames to define different simulation objectives. In addition, we may have many variations of a model based on the simulation and the objectives used. If we look at multiple System of Systems M&S’s that an engineer may build to perform a virtual test of all the configurations of its system of systems, we may end up with many Simulators and many Experimental frames that might be slightly different or may be vastly different. We may also end up with many variations of the models.

D. PROPOSED SOLUTION

Using an Agent Based Modeling (ABM) approach, we believe it is possible to develop an architecture for a software framework that allows the use of one Simulator and Experimental frame in an SoS simulation. The one Simulator would drive all the agents (models) together. The one Experimental frame would allow the user to set input parameters for all the agents (models) in the SoS simulation over many configurations. This architecture would then allow us to specify and initialize SoS metrics in one location—in the one Experimental frame, and observe changes in SoS metrics in one location—in the one Simulator.

It would also allow us to take it a step further. Using the definition of “swappable” to mean the ability to exchange one item for another, we would be able to

create swappable Simulators and swappable Experimental frames such that the agents do not have to change when the Simulator and/or the objectives of the experiment change. In addition, following this logic, we believe it is possible to reuse this Simulator and Experimental frame in multiple simulations without change. Because this allows for multiple swappable and reusable Simulators and Experimental frames, this software architecture allows for choosing the Simulator and/or an Experimental frame at runtime by the user to allow different simulation methods and objectives for each execution of the SoS M&S.

Our research is to architect a Simulator Exchange that allows one Simulator per simulation method (DEVS, DTSS, DESS) to be used by all SoS M&S in order to provide maximum reusability for all SoS simulations, without causing change to the other elements of the M&S. In addition, we intend to architect an interface to this Simulator such that Simulators are swappable. This means that we are not just tied to the three simulation types listed in Zeigler (2000). A programmer can create other types, such as human in the loop, etc., that have not yet been identified without changing the other elements of the simulation.

Because we can do this same exchange method with the Experimental frames, we will seek to design an Experimental frame Exchange that allows the reuse of an Experimental frame to be used by all SoS M&S. We will also allow these Experimental frames to be swappable. This will allow any objective to be chosen at runtime for an SoS M&S without changing any of the other elements in the simulation. Because we can now have multiple models interacting together and driven by one Simulator, we will call the collection of models the Component System Model Collection.

We believe that these exchanges will allow for the collection of SoS metrics in a Metrics Collector that will interface with the Simulator and Experimental frame in such a way that will enable the systematic identification and quantification of emergent behavior among the interactions of the agents (component systems). For the purpose of this research, emergent behavior will be defined as behavior of one component that is affected by another component. The Metrics Collector will drive the tasks of specification and collection of the metrics through the tasks of experimental data generation (by the

Experimental frame) and identification of changes to the SoS Metrics and interactions among the agents (by the Simulator) and further enable the reuse of the Experimental frames and the Simulators.

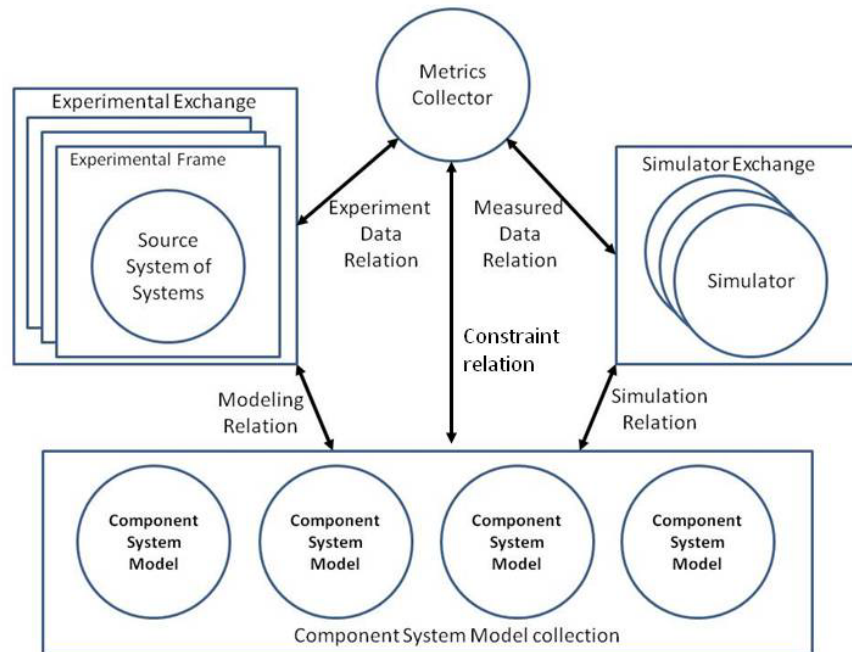
We are exploring the gathering of these SoS metrics in two ways. First, we allow each agent to identify the data an analyst wants to collect (through the Experimental frame) and alert the Metrics Collector when the values have changed using the publish/subscribe architectural style. Second, we begin with the method described in Chan (2011) for identifying emergent behavior. Chan increases a counter containing the number of interactions among agents whenever an agent has to interact with another agent based on some action that the agent has to perform. Chan then graphs these interaction metrics for each run and asserts that the interaction metric deviates from a normal curve (for a particular run) when emergent behaviors arise. Using M&S, these interactions can be found when one agent causes an event in another agent (such as one agent causing the destruction of another). We found that it is not just in the deviation but emergent behavior can be found in any run. We can identify emergent behavior in any run by overlaying the agent metrics with the interaction metrics for a run. These interactions would be identified in the Simulator, and analyzed and graphed (together with the SoS metrics defined in the model) in the Metrics Collector.

These two sets of data, the agent metrics and the interaction metrics between the agents, are graphed together to show the relationships among the data and among the agents. We believe that these relationships will lead us to identify and quantify emergent behavior.

We can collect all SoS metrics in the Metrics Collector where we can evaluate the data collected to determine if it can be identified as emergent behavior. This central collection point allows us to collect data across the System of Systems simulation.

Figure 6 is a depiction of how the Experimental frame Exchange, the Simulator Exchange, the Metrics Collector, and models (also known as agents) all interface together to produce an SoS simulation.

Figure 6. Modified M&S Theoretical Framework Entities and Their Relationships



Let's look at how this helps BMDS. The Simulator Exchange will allow the development of a BMDS simulation that will have one DEVS Simulator and one DTSS Simulator that can be interchanged without changing any of the weapon models. This interchange will allow one BMDS simulation to be driven by both Discrete Events and Discrete Time without any changes to the other elements. The Experimental frame Exchange will allow the selection of multiple objectives for the BMDS simulation without changes to the other elements. We can then collect agent metrics and a counter for interactions among agents for any type of BMDS simulation. This allows us to systematically collect and graph metrics that will lead us to identification and analysis of emergent behavior.

In order to have reusable and swappable Simulators and Experimental frames, we must first develop a framework that will allow this. We believe this framework has already been developed. The Orchestrated Simulation through Modeling (OSM) framework developed at Naval Surface Warfare Center, Dahlgren Division (NSWCDD) is such a framework. Winfrey, Baldwin, Cummings, and Ghosh (2014) tells us that the OSM framework allows Discrete Event System Specification (DEVS) Modeling and

Simulation (M&S) elements to be developed separately as plug-ins and combined with output visualization plug-ins and model plug-ins to form a complete simulation. After a swappable Simulator and Experimental frame and Metrics Collector are developed, we will “plug them in” to OSM, along with models (or agents) to build an SoS simulation as a proof of concept.

E. HYPOTHESIS

An SoS M&S framework architecture for system-of-system modeling and simulation can be developed to improve the process of identifying and quantifying emergent behavior, resulting in the

1. Reduction of effort a programmer needs to implement changes to an SoS simulation, and
2. Reduction of time an analyst needs to set up an experiment for an SoS simulation.

F. SPECIFIC GOALS OF PROPOSED RESEARCH

- To develop an architectural design of a Simulator Exchange that allows the swapping of Simulators within one SoS M&S or across multiple SoS M&S without changing any of the other elements of the SoS simulation
- To develop an architectural design of an Experimental frame Exchange that allows the swapping of Experimental frames that are swappable within one SoS M&S or across multiple SoS M&S without changing any of the other elements of the SoS simulation
- Because of the reusability of the Simulator and Experimental frame Exchanges, develop an architectural design of a Metrics Collector that allows the collection of SoS metrics that can be used to identify and quantify emergent behavior in an SoS simulation
- To demonstrate the ability to identify and quantify emergent behavior via the data collected by the SoS Metric Collector.

G. NEW CONTRIBUTIONS

- A new software architecture for a SoS M&S framework that enables efficient identification and quantification of emergent behavior in an SoS M&S. This architecture allows swappable/reusable Experimental frames and Simulators that can be used in one SoS simulation and across SoS simulations without changing the other elements in the SoS simulation. It

also allows the specification and collection of SoS metrics that enables the identification and analysis of emergent behavior in the SoS simulation. This architecture separates the specifying, collecting and processing of SoS metrics from the simulation code.

- An improved process for an analyst to identify and quantify emergent behavior. This is done by developing SoS M&S that separates four concerns of this type of simulation: (1) the specification, identification, and quantification of emergent behaviors via M&S metrics, (2) the specification, selection and instantiation of Experimental frames, (3) the specification, selection and instantiation of Simulators, and (4) the specification, selection, and instantiation of agents.

H. RESEARCH STRATEGY AND METHODS

1. Research Questions

- Can an architectural design for M&S framework components be developed such that they can together allow the identification and quantification of emergent behavior in an SoS simulation?
- Can models be written such that the model will not change when the simulation type is changed? In other words, can swappable Simulators be designed such that models do not have to change when a simulation type is changed (i.e., between discrete event and discrete time)?
- Can models be written such that the model will not change when the objectives of the SoS simulation are changed? In other words, can swappable Experimental frames be designed such that the models do not have to change when the SoS simulation objectives change (i.e., using stochastic values or using a lattice type experiment)?
- Can a Metrics Collector be designed so that it can be used to identify and quantify emergent behavior in an SoS M&S?
- What is swappability in regard to software components? How does it relate to reusability?
- Can the Simulator and Experimental frame be constructed such that they can interact with this Metrics Collector?

2. Research Approach and Methodology

This work will develop the following:

- Software architecture design for a swappable Simulator (known as a Simulator Exchange) that can be used for any type of simulation

- Software architecture design for a swappable Experimental frame (known as an Experimental frame Exchange) that can be used for defining any objectives for an SoS M&S
- Software architectural design for a Metrics Collector that allows the collection of SoS metrics used in the identification and quantification of emergent behavior in the interaction of the models in an SoS simulation
- Demonstration of the Experimental frame Exchange, Simulator Exchange, and Metrics Collector using a predator/prey simulation

I. DISSERTATION ORGANIZATION

Chapter II describes work done in areas related to this research. This research draws from many areas. This includes software architectures, M&S research, emergent behavior, and interaction metrics. We also studied and compared current M&S frameworks, languages, and toolkits to ensure that this research is improving the state of practice.

Chapter III describes the design methodology for each of elements of the framework described in this research. This includes the agents (that can also be thought of as the models), the swappable simulators, the swappable experimental frames, and the metrics collector.

Chapter IV describes the process for identifying and quantifying emergent behavior in an SoS simulation built using the framework described in this research. This chapter describes each step in this process.

Chapter V describes the proof of the contribution listed in Chapter I. It describes the Predator Prey simulation and the BMDS simulation developed using this research. This chapter shows evidence of how this research identifies and quantifies emergent behavior in the Predator Prey simulation. This chapter also shows how the BMDS simulation shows evidence of reusability of experimental frames and simulators.

Chapter VI is the conclusion. It provides the summary of significant findings, and it describes considerations for future research.

THIS PAGE INTENTIONALLY LEFT BLANK

II. BACKGROUND

This chapter describes work done in areas related to this research. This research draws from many areas. This includes software architectures, M&S research, emergent behavior, and interaction metrics. We also studied and compared current M&S frameworks, languages, and toolkits to ensure that this research is improving the state of practice.

A. SOFTWARE ARCHITECTURES

1. SoS Architectures

The elements that we are creating must be plugged into a framework in order to tie or link the Simulator Exchange, the Experimental Exchange, and the Metrics Collector to the models. Because we are creating a simulation of an SoS, the software architecture needs are equivalent to what is needed for an SoS. Selberg and Austin, (2008) tells us that an SoS needs a decentralized architectural paradigm. Two key characteristics of this paradigm are emergence and evolution. Emergence is defined as the properties which do not belong to any of the individual components but can emerge from the SoS. Evolution is the change over time that occurs in the SoS as component systems are added, removed, or replaced.

Selberg and Austin, p. 3, (2008) identifies two principles that must be met in order to allow the SoS to evolve as the component systems are added, removed or replaced. These are:

- The complexity of the system of systems framework does not grow as constituent systems are added, removed, or replaced.
- The constituent systems do not need to be re-engineered as other constituent systems are added, removed, or replaced.

In order to meet these two principles, the SoS framework must be loosely-coupled and have standard interfaces and interface layers. The standard interfaces and interface layers are to insure that the component systems do not have to be changed as components are replaced, added or removed. As part of our research, we will define standard

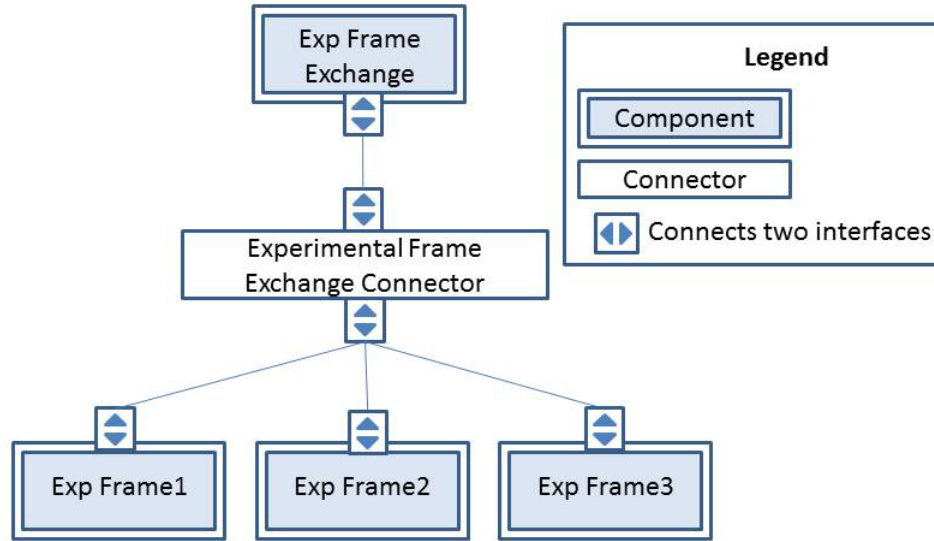
interfaces and interface layers in our architecture to ensure that we meet the two principles defined by Selberg and Austin.

2. Components and Connectors Architectural Style

Components and connectors (C2) provide interoperability between components of different types (i.e., developed with different programming languages). Taylor, Medvidovic, and Dashofy (2010) tells us that it was created to allow applications to work with new platforms by extracting the components from calls to platform services such as operating systems and user interfaces from the body of the components that need these services. It also allows for substituting one component for another to achieve applications that are similar but still different.

Applications which use the C2 style are layered networks of concurrent components connected by message routing connectors. In this style, there is no direct component to component communication, as shown in Figure 7. It only allows the connections of components to connectors. This enforces layering which promotes the ease of modifying an application to work with different platforms, and promotes the ability to swap components. C2 uses domain translation to make this happen. Domain translation is the transformation of requests from a component into a form that is understood by the recipient of that request. This principle is also used for the notifications provided by a component.

Figure 7. Example of C2 Style



The C2 style manages the flow of control and flow of data for an application. The flow is managed by using one or more channels to link the interacting components. This control and data flow will be used to determine and modify the timing of an event (convert the event to a timed event) being sent to the event driver of the Simulator.

The C2 style provides conversion services to transform the data involved in an interaction to be that required another component. These type of connectors are called adaptor connectors. In our case, we will convert discrete events to discrete timed events.

We will use the C2 style to allow a DEVS Simulator to drive DTSS models (agents) and a DTSS Simulator to drive DEVS models (agents). The benefit of extracting components from calls to platform services allow us to switch between various simulation types without having to change the agent (model) itself. The benefit of substituting one component for another is feature that allows us to develop a Simulator that can convert between Discrete Events and Discrete Time without changing the models themselves.

This architectural style will be used in this work to connect: Simulator Exchange and Simulators; and Experimental frame Exchange and Experimental frames.

3. Publish/Subscribe Architectural Style

The Publish/Subscribe software architectural style is used to allow the sending of information (via messages) without having to program each message passing between elements. In this architectural style, the subscriber registers to receive information; the publisher maintains the list of subscribers. The publisher broadcasts the information (via messages) to the subscribers either synchronously or asynchronously (Taylor, Medvidovic, & Dashofy, 2010). This architectural style will be used in this work for passing metrics data: between the agent (publisher) and the Metrics Collector (subscriber); and between the Simulator Exchange (publisher) and Metrics Collector (subscriber).

B. MODELING AND SIMULATION THEORY BY ZEIGLER

Zeigler (1976) describes a Modeling and Simulation (M&S) formalism and framework for each of the three simulation methods. They are:

- “DEVS—Discrete Event System Specification”
- “DTSS—Discrete Time System Specification”
- “DESS—Differential Equation System Specification (also known as Continuous Time)”

Table 1 shows the differences in these three simulation methods, as taken from Zeigler (1984).

These formalisms are defined by three objects:

- Experimental frame—defines the objectives of the model
- Simulator—drives the simulation by passing events and time to the models
- Model(s)—each model describes a real world object

The models can be coupled together to form collections of models and these can be hierarchical in nature.

Table 1. Differences in Simulation Methods

	Differential Equations	Discrete Event	Discrete Time
Time Base	Continuous Reals	Continues Reals	Discrete Integers
Sets - Inputs, Outputs, and States	Real vector space	Arbitrary	Arbitrary
Input segments	Piecewise continuous	Discrete event segments	Sequences
State and output trajectories	Continuous segments	Piecewise constants/segments	Sequences

After Zeigler, B. (1984). Multifaceted Modelling and discrete Event Simulation. London: Academic Press.

1. Experimental Frame

The Experimental frame contains a behavior database which is a collection of gathered data from the source system. This data is used in defining the conditions under which the system is observed. This data is a subset of all the data in the source system. There may be multiple Experimental frames for one source system, defining different objectives for the simulation of that source system.

According to Zeigler (2000), there are two interpretations of the Experimental frame. One interpretation is to use it to define the types of data elements that will go into the behavior database. The second interpretation is to use the Experimental frame “as a system that interacts with the source system to obtain desired data.” In our research, we will use the first interpretation of the Experimental frame to define a Graphical User Interface (GUI) to offer all the data in the behavior database to the user for changing.

2. Simulator

The Simulator (according to Zeigler) is a set of instructions used to execute the model and generate behavior. (Zeigler, 2000) states that the Simulator and the model are separated to allow the model to have multiple Simulators.

a. DEVS

Zeigler (1976) tells us that “the Discrete Event System Specification is a formalism for modeling and analyzing discrete event systems.” DEVS can be viewed as a finite state automaton where each state has a lifespan, the outputs are determined by the state alone, and coupling operations together in a hierarchical fashion. In Zeigler (2000), the DEVS formalism is defined as:

$$\text{DEVS} = (X, S, Y, \delta_{\text{int}}, \delta_{\text{ext}}, \lambda, \text{ta})$$

X = set of inputs

S = set of states

Y = set of outputs

$\delta_{\text{int}} : S \rightarrow S$ = internal transition function, could be because of time

$\delta_{\text{ext}} : Q \times X \rightarrow S$ = external transition function, comes from outside the model

$Q = \{(s, e) \mid s \in S, 0 \leq e \leq \text{ta}(s)\}$ is the total state set

E = time elapsed since last transition

$\lambda : S \rightarrow Y$ = output function

$\text{ta} : S \rightarrow \mathbb{R}_{+0, \infty}$ = time advance function

A discrete event Simulator uses a queue to hold events. This queue is ordered by time of execution of the events. The events can be created, cancelled, or rescheduled. Each event in the queue contains an event handler that is executed when the event is performed. There is an efficiency advantage to discrete event simulation because state transitions are only evaluated when an internal or external event is received.

There is an ordering disadvantage to discrete event simulation because of simultaneous events. There is not a rule on how to handle events that are affecting agents (models) at the same time. For example, in a Predator Prey simulation where wolves eat rabbits and rabbits eat grass, if a wolf moves toward a rabbit that moves toward grass, the wolf's movement is affected as to whether the rabbit moves before or after the wolf. This

could change the simulation results. Therefore, the results are depending on the ordering of the activation built into the Simulator (Zeigler, 2000).

The DEVS formalism is closed under coupling because of the result of the network of systems is itself a system specified in the DEVS formalism (Zeigler, 2000).

b. DTSS

Zeigler (1976) tells that “the Discrete Time System Specification is a formalism for modeling and analyzing systems with a discrete time base.” In Zeigler (2000), the DTSS formalism is defined as:

$DTSS=(X, Y, Q, \delta_{int}, \delta_{ext}, \lambda, c)$

X = set of inputs

Y = set of outputs

Q = set of states

$\delta : Q \times X \rightarrow Q$ = state transition function

$\lambda: Q \rightarrow Y$ = Moore-type output function

Or

$\lambda: Q \times X \rightarrow Y$ = Mealey-type output function

c is a constant representing the time base

There is an efficiency disadvantage to discrete time simulations because state transitions are evaluated at every time step. This means that if a simulation has 500 agents (or models), the simulation would have to evaluate each for a possible state transition at every time step (say every second).

There is an ordering advantage to discrete time simulations because all state transitions occur before any output is processed. For example, in the Game of Life simulation, in which each cell is either considered living or dead based on the number of living neighbors that it has, all cells are evaluated at a time step and then the cells are changed. Otherwise, a cell’s evaluation of its neighbors may change midstream causing a change in the results.

c. DESS

Zeigler (1976) tell us that “the Differential Equation System Specification is a formalism for modeling and analyzing systems with a continuous time base.” In Zeigler (2000), the DTSS formalism is defined as:

$DESS=(X, Y, Q, f, \lambda)$

X = set of inputs

Y = set of outputs

Q = set of states

$f: Q \times X \rightarrow Q$ = rate of change function

$\lambda: Q \rightarrow Y$ = Moore-type output function

Or

$\lambda: Q \times X \rightarrow Y$ = Mealey-type output function

3. Model

According to Zeigler (2000), the model in the simulation is “a set of instructions, rules, equations and constraints for generating input and output behavior.” In other words, models change states and use output mechanisms in answer to inputs (that are referred to as input trajectories since they happen over time). These models generate outputs (that are referred to as output trajectories since they happen over time) using the output mechanisms. These outputs are based on the initial state of the model. Models can be coupled together in a hierarchical fashion to form larger, more complicated models.

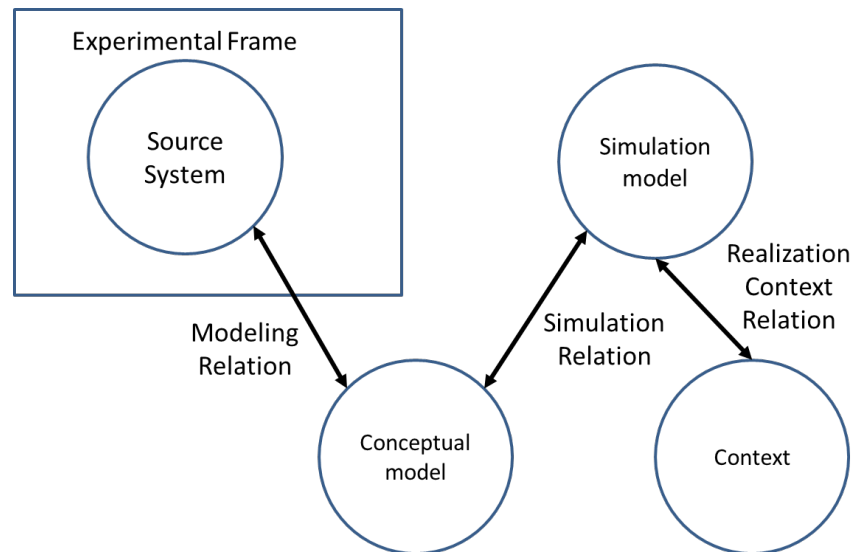
Models can be atomic, coupled or hierarchical. An atomic model is the most basic. It is a model than cannot be decomposed any further. A coupled model is one that contains atomic models connected (or coupled) together. These connectors can either be for connecting an external source to a model for input, connecting a model to an external element to accept the model’s output or for connecting one model’s output to another model’s input. A hierarchical model is a model coupled with other models that may be either atomic or coupled.

C. REUSABILITY OF DEVS COMPONENTS

Yilmaz (2004) describes model reuse with the DEVS formalism by representing and storing the context information of the models and relating this information to the model developer’s intentions. This work touches on how the Experimental frame and the Simulator can be reused. This work uses the term “simulation model” to represent the model with its Simulator, again emphasizing that each model has its own Simulator to drive it. This work defines the notion of context objects, and these objects are defined

independently from the simulation model. Figure 8 shows the connection between objects in the simulation to allow model reusability.

Figure 8. Adding Context to the Models to Allow Model Reusability



From Yilmaz, L., & Oren, T. (2004). On the need for contextualized introspective models to improve reuse and composability of defense simulations. *The Journal of Defense Modeling and Simulation: Applications, Methodology, Technology, Vol 1 (3)*, pp. 141–151.

Spiegel (2005) takes this work a step further and defines the model context in terms of the Experimental frame (or objectives of the experiment). This work defines “the context of a model as the set of all Experimental frames under which the model is valid.” This means that a model is defined in terms of all the objectives that can possibly be used in executing this model. This work also looks at how validation constraints impact model reusability.

D. MODELING TYPES

Parunak, Savit, and Riolo (1998) defines the similarities and differences between Agent Based Modeling (ABM) and Equation Based Modeling (EBM). Both modeling approaches “simulate a system by constructing a model of the system and then executing it on a computer.” Both modeling approaches focus on the individual items to model and the measurable characteristics of interest known as the observables. ABM is made up of

agents that encapsulate behaviors, while EBM is a set of equations. In the execution of these models (the simulation), ABM execution is the emulation of the behaviors of the agents while the execution of EBM is the evaluation of the equations.

The differences between the two approaches include the relationships among the entities that are modeled and the level each approach focuses on. For EBM, the equations express the relationships among the observables. ABM focuses on the interactions among the entities. For the latter difference, EBM focuses on the system level observables, while ABM defines behaviors at the individual agent level and observes what happens when these individual agents interact. Because of these differences involving the interactions of the entities, we feel that ABM is the more appropriate modeling approach to use for System of Systems M&S.

E. AGENT-BASED MODELING AND SIMULATION

Macal and North (2008) describes agents as having the following features:

- An agent's behavior must be adaptive.
- Agents must be capable of making independent decisions.
- Agents are self-contained, autonomous, and self-directed.
- Agents interact with other agents.

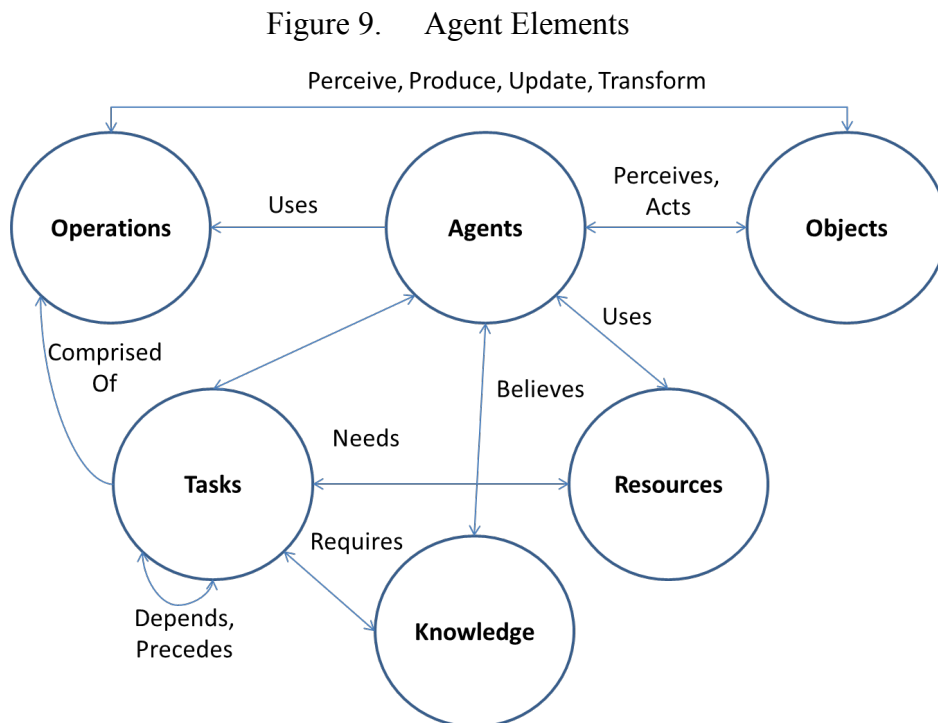
These features provide a good forum for producing emergent behavior as the agents interact with each other and make independent decisions of what to do next. The importance of emergence is complex behavior arises without complex rules.

Yilmaz and Oren (2009) tells us that an agent is autonomous and flexible to be both reactive and proactive. It perceives its environment and acts on it. Yilmaz also tells us that a system of agents in modeling contains the following elements:

- "An environment that envelops the agents, objects and resources"
- "A set of objects that are perceived and acted on by agents"
- "A set of agents that represent active entities"
- "A set of resources that are used by agents to perform tasks"

- “A distributed knowledge base used by each agent to define beliefs about the environment, an agent itself and the other agents in the system”
- “A set of tasks that are the actions of the agents”
- “A set of operations that allow the agents to perceive, produce, transform and manipulate objects”

Figure 9 is a diagram of the agent elements.



From Yilmaz, L., & Oren, T. (2009). *Agent-Directed Simulation and Systems Engineering*. Weinheim, Germany: Wiley-VCH.

Wooldridge and Jennings (1995) claims that agent systems are really a software engineering paradigm, not a modeling paradigm. This paper provides a method for designing and implementing agents using the object oriented programming approach as a basis for developing agents.

Chan (2011) states that agent based simulation is different from other modeling approaches because of its ability to simulate interacting autonomous agents. These interactions allow for emergent behaviors to appear.

Wijesekera, Michael, and Nerode (2005) developed a framework called BMD-Agents (BMDA), used to model distributed control between communicating agents that must work together to destroy threat missiles. This framework models the Command and Control (C2) of the BMDS. The agents that execute the C2 processes are driven by doctrine, and the organizational structure chain of command (also known as combination of agents) is driven by C2 policy. The definitions of event-condition-action rules and command and control structures can be used in our work as a basis for modeling the BMDS communications.

F. EMERGENT BEHAVIOR

Emergent behavior is not a term connected only with computer science. Blitz (1992) tells us that the word was first used by G. H. Lewes in his book, *Problems of Life and Mind* (1874–1879) in discussing the evolution of the human mind. Lewes contrasted resultants and emergents. He stated that emergents cannot be reduced to the sum or difference of their parts. Corning (2002) brings forth that the work in genetics in the 1920s and 1930s . It brought an analytical experimental approach to biological work and brought reductionism (vice emergence) in science. Corning tells us that reductionism scientists argued that in a system of components “the properties of the complex can be inferred from those of the components.” “Emergence re-emerged when there was growth in the scientific interest in complexity and the development of non-linear mathematical tools.”

Corning (2002) tells us that with this re-emergence came different views of emergence. Some thought that the whole system is bigger than the components that make it up. Others had a more reductionist view and felt that much comes from little. Still others felt that emergence is part of the “coherent structures, patterns and properties that arise during the self-organization in complex systems.”

Macal and North (2008) states that when the behavior of an agent affects other agents, and those agents’ behaviors ultimately affect the original agent, a complex feedback dynamic is put in motion that produces emergence. Li, Sim, and Low (2006) states that software agents cannot act like people due to speed of their decisions, lack of

flexibility, and learning difficulties. Therefore, they can act in unexpected ways. This unexpected behavior cannot be found in the model's specification, because it is an outcome of the interactions among the agents. We can then deduce that if we cannot define or specify emergent behavior, we must use some other method to analyze it.

Privosnik, Marolt, Kavcic, and Divja (2002) describes the thought of specification by explaining that “particular agents do not need to be individually complex for the system to demonstrate complex emergent behaviors.” Its global behaviors define the characteristic of an emergent system, which are the result of interactions among many “relatively simple parts, and these cannot be predicted simply from the rules of those underlying interactions.”

Gore, Reynolds, Tang, & Brogan (2007) describes research in allowing “a subject matter expert to test a hypothesis about emergent behavior.” Their work builds on this to develop an approach to observe and test simulation behavior while allowing “the users to validate or reject emergent model behaviors.” This work ties “with the DEVS formalism by validating emergent behavior for a given set of conditions or DEVS Experimental frames (also known as objectives of the model).” This work uses “causal inference procedures to reveal interactions of known abstractions in the model, which will cause emergent behavior.” “Causal inferencing finds cause and effect relationships among observed variables.” This causal inferencing explains or describes a set of observations (Gore and Reynolds, 2008). His work relies on abstracting portions of the model to reduce complexity and focuses on the model during implementation.

Samaey, Holvoet, and De Wolf (2008) analyzes emergent behavior in self-organizing systems using an “equation-free macroscopic analysis” method. In this method, coarse time steps are made through the model using short simulations (parts of the simulation of the model, not the entire model at one time). The critical parts of this method are to define the set of macroscopic variables that characterize the self-organizing emergent behavior and to initialize the system given only macroscopic variables. This paper focuses on establishing and initializing the macroscopic variables. This method works well for swarming, which is a type of self-organizing system, but not for systems where there is a complicated rule set.

Boschetti, Prokopenko, Macreadie, and Grisogono (2005) tells us that patterns at the SoS level emerge only from interactions among the component systems acting on rules which “are executed using local information, without regard to the global pattern,” but this pattern is not displayed in a component system alone.

G. INTERACTION METRICS

Chan (2011) tells us that in order to be emergent behavior, it must occur from the interaction of the components. Interaction can be defined as any action that occurs between two or more components and has an effect on those involved components. Chan states that this interaction always causes a state change in high fidelity models. Our research contends that we can also always name this interaction emergent behavior in low fidelity models as well.

Chan (2011) has investigated the use of metrics to identify emergent behavior. In his work, he purports to increase a counter containing the number of interactions among agents whenever an agent has to interact with another agent based on some action that the agent has to perform. Chan then graphs these interaction metrics and asserts that the interaction metric for a run deviates from a normal curve when emergent behaviors arise. In a study of three models, this hypothesis held true for two of the models. This work is new and needs to be further studied.

Chan states that there are two types of interactions. They are regular interactions and effective interactions. Regular interactions show when “an agent initiates or receives contact with another regardless of whether this action induces any outcome.” Effective interactions are regular interactions with a final outcome.

Chan provides pseudo code for these interactions but states that the measurement may be different in different models because the interactions are different. In our research we have developed a consistent method for defining those interactions through swappable Simulations and Experimental frames.

In our research we include these interaction metrics with agent metrics on a single graph. Instead of showing the deviation for a set of runs, we show emergent behavior in a

single run by identifying the changes in slope of the interaction metrics as it occurs at the same time as the changes in the agent metrics.

H. M&S INTEROPERABILITY

Tolk, Diallo, Padilla, and Turnitsa (2011) tells us that while there are successful areas of interoperability in software (such as cloud computing and web services), the ability to plug and play models in a simulation has not yet been realized. The challenge is to ensure the logical equivalence of all representations of some real object in the federation (or grouping of objects/models). The problem here is modeling the dependence of one object on another. To further complicate the matter of interoperability, it is now believed that perception must now be taken into account when determining if two models can operate together. The underlying issue here is that in other areas of science interoperability refers to making two objects interoperate, but in M&S interoperability refers to making two simulations or representations of objects interoperate. The answer lies in the conceptualization phase of these models. We have to understand the concepts that were derived from the objects we are simulating in order to determine how these simulated operate together. We must ensure that concepts that are common in the interoperating simulations must be represented consistently.

I. MODELING AND SIMULATION SOFTWARE FRAMEWORKS AND LANGUAGES

In order to use swappable Experimental frames and Simulators, we must find a software framework or modeling language to allow the use of these. This framework will be for use by computer scientists to build robust SoS simulations. We have researched five such products.

1. Frameworks

a. Orchestrated Simulation through Modeling (OSM)

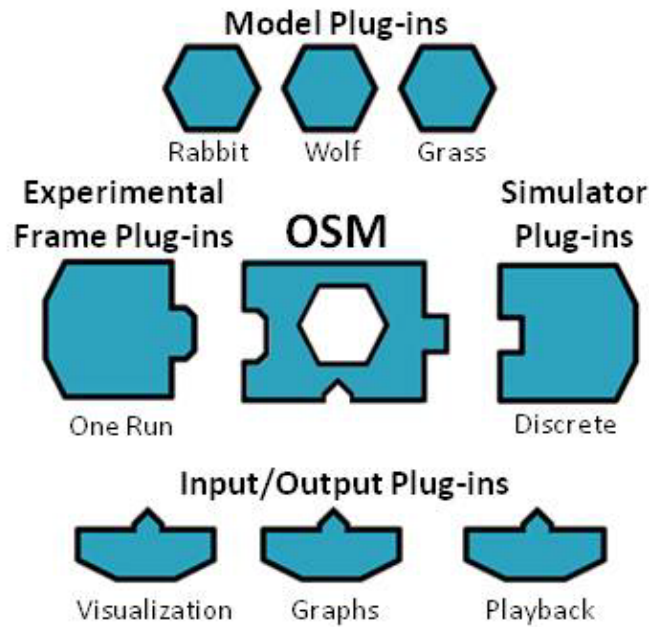
OSM is a government-owned M&S framework, developed by Naval Surface Warfare Center Dahlgren Division in the Strategic Software department. This framework is model agnostic, domain agnostic and enables the integration and execution of multiple

independently developed models. It uses plug-ins as a method of plugging all the components together. It facilitates the development of a rich set of re-usable plug-ins. It can be used for any type of M&S project because of its model agnostic methodology. It allows the sharing and tying together of models from various organizations to create an SoS simulation. OSM is patent pending, and is free to anyone in the Department of Defense. It allows the development of agents from scratch or can import existing models by wrapping the models to produce agents from the inputs/outputs of the models. This works best if the models were developed to interact or provide data throughout the simulation.

Winfrey, Baldwin, Cummings, and Ghosh (2014) tells us that the OSM framework allows “Discrete Event System Specification (DEVS) modeling and simulation (M&S) frames to be developed separately as plug-ins and combined with output visualization plug-ins and model plug-ins to form a complete simulation.” This framework “allows input plug-ins (model, Experimental frame), execution plug-ins (Simulator), and output plug-ins (visualization of output on a world map, graphs) to be developed separately and pieced together to form a unique system while allowing the development to be compartmentalized.” Figure 10 shows the OSM software architecture. Each element is plugged into the OSM framework to create an SoS simulation.

OSM was built to have computer scientists as users. It is written in Java, and has many reusable methods, classes and interfaces for doing a variety of M&S, such as displaying on any type of map or displaying any type of output (known as output agnostic). It also was built for any type of agents. In other words, it was not built for one type of domain (such as Aegis Weapon System, Army Weapons, etc.). We call this “model agnostic.”

Figure 10. OSM Software Architecture



From Winfrey, C., Baldwin, B., Cummings, M. A., & Ghosh, P. (2014). OSM: An Evolutionary System of Systems Framework for Modeling and Simulation. *Proceedings of the Spring Simulation Multi-Conference*. Tampa.

One such reusable class that it provides is the Bulletin Board class. This class provides the ability to publish and subscribe to data. This is the class by which we will provide the metrics data to the Metrics Collector.

b. NetLogo

Wilensky (1999) tells us that NetLogo is a programmable modeling environment, and available for free. It was developed to simulate natural and social phenomena. It is a good environment for developing agents in a classroom setting. NetLogo was developed for many types of users, from children in school to non-programming domain experts (Kornhauser, Rand, & Wilensky, 2007). Because of its users, it uses the terms turtles, patches, and observer as programming terms for developing agents. It was authored by Uri Wilensky in 1999 and “continues to be developed at the Center for Connected Learning and Computer-Based Modeling.”

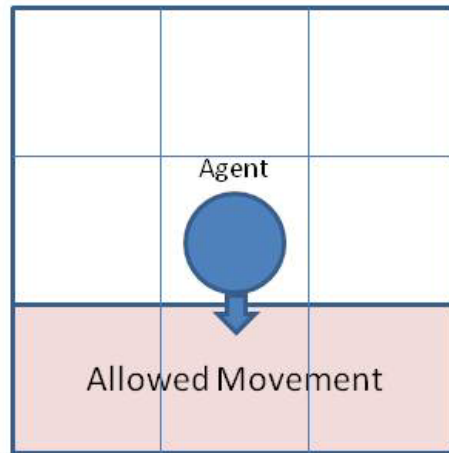
NetLogo allows for “the modeling complex systems over time.” “Modelers can give instructions to many agents all operating independently, but interacting.” This allows the exploration of behaviors between the agents and the patterns of behavior that result.

NetLogo comes with a set of examples that include the description, the code itself and the Graphical User Interface that lets the user manipulate the input or experiment parameters to change the outcome. Wilensky (1999) tells us that “these examples model the natural and social sciences including biology and medicine, physics and chemistry, mathematics and computer science, and economics and social psychology.”

NetLogo has its own language. This does not look like other programming languages and is hard for programmers to understand. This would cause programmers to have to learn a new programming language.

We attempted to match an SoS simulation developed with the OSM framework with the same application built in NetLogo. We found it impossible to do because there seems to be undocumented and difficult to understand/reproduce features in the NetLogo methods. An extremely tiny alteration seems to make a major difference. Just changing from random movements to directed movements made for very different results. For example, using a grid of nine cells surrounding an agent (see Figure 11), the NetLogo version directed the agents to move ahead one step within a 50 degree area to the left and right of the direction of the agent. In our OSM example, using this method and allowing movement to any of the nine cells surrounding the agent, caused very different results. This led us to believe that NetLogo provided methods must have more functionality involved than documented.

Figure 11. Movement of Agent in NetLogo's Predator Prey Example



NetLogo does not list in its documentation the ability to change its simulation type (discrete event, discrete time, etc.) without changing the model itself. It was clear from the examples given that to change from a discrete event to discrete time driven simulation would mean a change to the models.

c. Simulink through MATLAB

MATLAB, The Language of Technical Computing (2015) tells us that MATLAB (which stands for MATrix LABoratory) “is a high-level language and interactive environment for numerical computation, visualization, and programming.” It is used by engineers in both academia and industry “to analyze data, develop algorithms, and create models and applications.”

MATLAB is a commercial product and can be expensive. The different versions of MATLAB may not backward compatible so the user (developer) must have the correct version on his platform. Historically, some interfaces to external code have changed from one revision of MATLAB to the next without backward compatibility. When this occurs, it requires the engineer’s code to be updated to match the latest MATLAB specification.

Simulink is a product built on MATLAB to simulate models, and must have MATLAB present to run. It is the tool most often used by engineers to develop simulations. This must be emphasized—it is used by engineers. It is not advertised as an

environment for computer scientists/programmers. That is because it has its own language. This language is proprietary to MATLAB and was written for the engineering community. Its unique structure is very different from programming languages familiar to programmers.

MATLAB, The Language of Technical Computing (2015) tells us that Simulink simulates block diagrams through the use of a graphical block diagramming tool. “It provides a set of predefined blocks that can be used and combined to create a detailed block diagram of a complex system.” Customized functions can be built “by using these blocks or by incorporating hand-written MATLAB, C, Fortran, Ada, or Java code into a model.” Additional predefined blocks can also be purchased.

Using the 2014 quoted prices on the MATLAB website, the cost of Simulink with MATLAB is over 5,000 dollars for one license. Each additional predefined block is about 2000 dollars a block. For example, if you were working on an aerospace project, you would buy MATLAB, Simulink and the aerospace predefined blocks. This will cost over 7,000 dollars for one license.

A toolbox has been created on Simulink that allows the simulation of hybrid systems. Sanfelice, Copp, and Nanez (2014) tells us that a hybrid system, for this context, is defined as “a dynamical system with continuous and discrete dynamics.” This means that it can simulate a system that has both continuous time and discrete event elements. An example this paper provides is a bouncing ball. The ball’s movement produces a constant movement in time but the hitting of the floor is an event. Sanfelice, Copp, and Nanez (2014) also states that this toolbox “uses the MATLAB language and the Simulink basic building blocks that define the dynamics of a hybrid system.” This toolkit provides an integrator system for integrating the different systems together, and postprocessing/plotting to show the results.

d. Systems Tool Kit

Systems Tool Kit (STK) is a physics-based software tool from Analytical Graphics, Inc. It was created to perform analyses of satellites but now is used to simulate ground, sea, air, and space assets as one integrated simulation. It was developed in 1989.

According to Systems Tool Kit (n.d.), it is built on a time-dynamic physics-based geometry engine. It uses physics vice agents as its modeling paradigm. It was built for a particular domain. Because of these two elements, it will not work well as a framework for our research.

e. Ptolemy

Ptolemy is a simulation and rapid prototype environment developed at the University of California Berkely in the Department of Electrical Engineering and Computer Science. According to Buck, Soonhoi, Lee, and Messerschmitt (1994) it was developed for heterogeneous systems. Its emphasis is on the simulation of networks, hardware with embedded software, circuits, and processors.

Eker, et al. (2003) tells us that the ability to be heterogeneous is through the use of directors. There is a director for each model. Directors have been developed to support discrete events and continuous time, along with others. Because of this allowance of discrete event models with continuous time models, hybrid system simulations can be developed with this framework.

One unique element of this framework is that it uses a hierarchical heterogeneous (dividing a complex model into a tree of nested heterogeneous models) style instead of the traditional amorphous style. The amorphous style allows different interaction mechanisms to be specified among the models. This can lead to unintended interactions (defined as emergent behavior in their work). The hierarchical heterogeneous style constrains the submodels at each level in the tree to be homogeneous; thus constraining the interactions between the submodels in one level. This reduces their definition of emergent behavior.

Ptolemy's building block is an actor. It seems comparable but different from agents. An actor does not have to be autonomous. It has an Initialize phase through which the user provides inputs. This is done in a simplistic manner, which means that it does not allow for all manner of input parameters. In addition, unlike the rest of Ptolemy, this GUI is not object oriented thus making it harder to understand. This initialize phase can be compared to the Experimental frame in Zeigler's theoretical formalism.

Eker et al. (2003) tells us that Ptolemy has a scheduler for each model. This scheduler can be compared to a Simulator in Zeigler's theoretical formalism. And its one-to-one connection between a scheduler and a model is in direct comparison to Zeigler's view of having one Simulator per model.

Two differences to OSM stand out. First, unlike OSM, Ptolemy was not developed for behavioral modeling. This is a further reason to infer that an actor is not an agent because agents are a good use of behavior modeling. Second, OSM uses one simulator which causes homogeneous modeling. Ptolemy's use of a director per model causes heterogeneous modeling.

2. Modeling Languages

a. Systems Modeling Language (SysML)

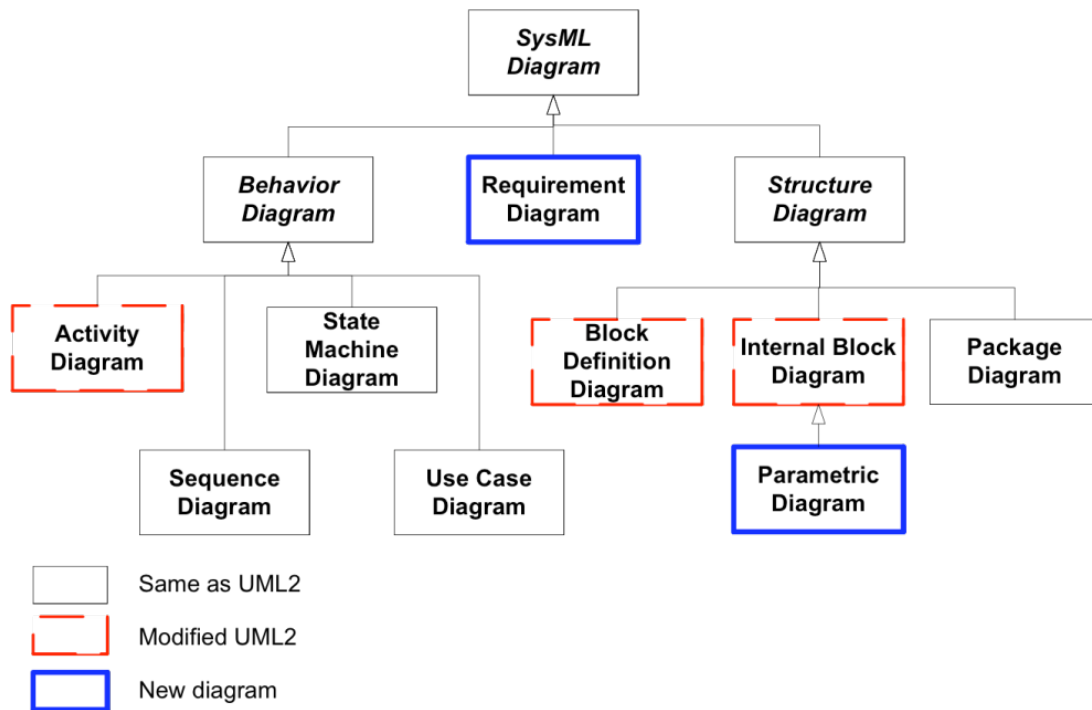
SysML Open Source Specification Project (n.d.) states that "SysML is a general purpose visual modeling language" developed for engineers and used for systems engineering projects. It emphasizes visual modeling principles. From this website we also know that in order to use SysML, two steep learning curves must be overcome:

- Becoming fluent in the language
- Learning how to work in a SysML Modeling Tool, many of which do not have easy to use user interfaces

SysML is used for Model-Based Systems Engineering (MBSE). MBSE is the use of visual modeling principles for system engineering activities through the system development life cycle. This engineering emphasizes multiple views as the primary work artifact throughout this life cycle. This also goes hand in hand with the system architecture model. The open standards that SysML relies on are used to specify the System Architecture Model. These principles ensure that the architecture model is architecture centric and expect the diagrams to maintain structural and functional relationships among each other that create traceability among all the views.

These commercial tools range from 300 dollars to over 10,000 dollars for one license. This does not include training and consulting fees. Figure 12 shows the diagrams that are for use in the SysML language.

Figure 12. SysML Language Diagrams



From *SysML Open Source Specification Project*. (n.d.). Retrieved July 25, 2015, from SysML.org: <http://sysml.org>

There is no test suite that allows the testing of SysML standard compliance in a tool. This means that care must be taken in choosing a tool. There are also a couple of open source SysML tools available (Modelio and Papyrus). According to SysML Open Source Specification Project (n.d.), these tools do not, at this time, compete with the commercial tools available on the market.

SysML is a dialect of the Unified Modeling Language (UML). It is a static language except when written in a tool (such as Enterprise Architect or MagicDraw) which allows the automation or execution of the language through the tool. It can be read by tools (such as Enterprise Architect or Magic Draw) which can then produce simulations of the models. This language must be learned and it is very different from programming languages like Java.

b. High Level Architecture (HLA)

Dingel, Garlan, and Damon (2002) tells us that HLA was developed in 1995 by a team of representatives from government, academia, and industry. It is a specification for interoperability among heterogeneous simulations (Kim & Kim, 2005). It provides a Runtime infrastructure and a common architecture for M&S. It allows for federated simulations to be integrated together, using this infrastructure. The term “Federate” is defined as being formed together into a single unit. To go with this, federates (a noun defined as an independently developed simulation component) are brought together to form a federation. A federation in the M&S world is a set of independently developed simulations formed together to create a single simulation.

HLA does not support simulations which need multiple federations (Kim & Kim, 2005). This does not allow modeling complex systems with hierarchical component models. This problem is one of the things that Zeigler’s theoretical formalism solves. Kim and Kim (2005) tells us that DEVS demonstrates how to model and simulate complex systems with hierarchical structures.

Dingel, Garlan, and Damon (2002) tells us that HLA contains services (or functions) that allow these federated systems to join together. This is known as a federation. HLA defines services (or functions that allow these federated systems to communicate. HLA also provides timing.

Dingel, Garlan, and Damon (2002) tells us that these services can be provided in C++ or Java. “There is no standardized protocol.” All the federated systems in a federation must use the same library of services in order to operate together.

HLA was developed almost 20 years ago. It is known to be complex, and its complexity has caused it to be updated over the years.

Dingel, Garlan, and Damon (2002) tells us that “the High Level Architecture (HLA) for distributed simulation defines a framework for the integration of cooperating, distributed simulations, possibly built by many vendors,” but all must be written in HLA. An HLA simulation/model can be plugged into the OSM framework, along with other simulations/models written in HLA or other methods, such as DIS (Distributed

Interactive Simulation) or TENA (Test and training ENabling Architecture), in order to build an SoS simulation.

J. SUMMARY

This chapter describes the many works used as a basis for this research. This research used the principles of Selberg's 2008 paper on evolutionary SoS architectures to determine the type of framework needed for our components. We found that the OSM framework provides this type of architecture. To design the components to plug into this OSM framework, we used the C2 architectural style and the Publish/Subscribe architectural style.

This work uses Zeigler's M&S theory to separate a simulation into an Experimental frame, a Simulator and models. We then expand on this to create swappable Experimental frames and Simulators. The Simulator Exchange allows us to choose between Zeigler's DEVS and DTSS theoretical formalisms for simulator types. In future work we (or others) will allow the ability to swap in a DESS simulator type as well.

Zeigler defines the Experimental frame as where the objectives are set. Our Experimental frame Exchange does this by providing the Graphical User Interface (GUI) for presenting each agent's experiment or scenario data to the user. It also provides the GUI for presenting the overall simulation experiment or scenario data to the user.

We used Yilmaz et al. and Tolk et al. work as a start for component reusability in a simulation. Parunak et al. and Macal's work allowed us to look at the differences between ABM and EBM. We determined ABM is best for our work.

We used the many emergent behavior articles to understand the various views/work on emergent behavior. We did this to determine how best to identify and quantify this type of behavior. Chan's work was used as a basis for creating metrics for showing interactions between agents.

We surveyed many frameworks to choose the best one for this work. OSM was chosen as our framework because it meets Zeigler's M&S theory and has an evolutionary

software architecture. NetLogo and Simulink were not chosen because of their use of languages is not familiar to computer scientists. STK was not chosen because it was built for a certain domain, and it is physics based. Ptolemy was not chosen because the use of multiple directors and schedulers in a simulation.

We looked at two modeling languages that have been used to develop models. SysML is used for MBSE and is used by engineers, not computer scientists. HLA is an older language that was used by many, but it does not follow Zeigler's theory for hierarchical component models, which are needed for SoS simulations.

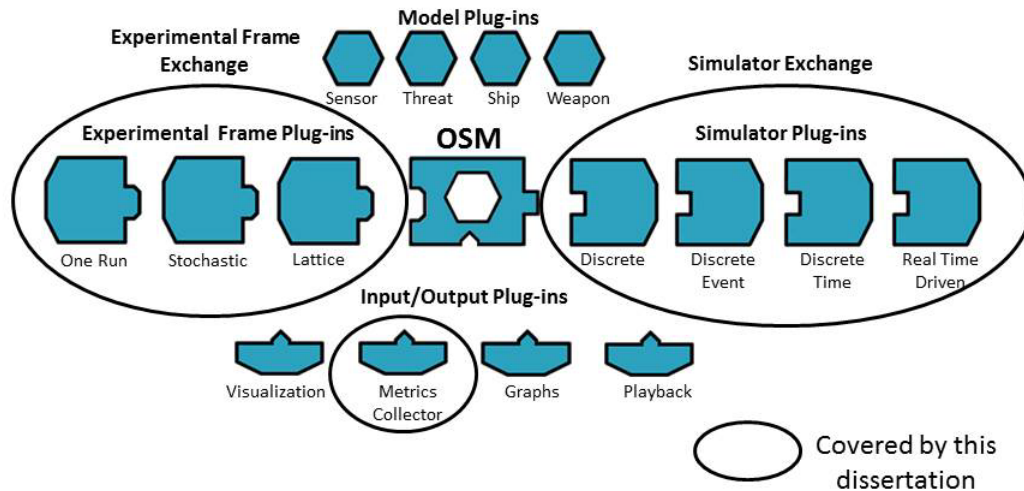
III. METHODOLOGY

We have developed a software architecture that enables the identification and quantification of emergent behavior in an SoS simulation. This architecture uses two architectural styles:

- Publish/subscribe to be used for the metrics data when set/changed
- Component and connector to be used for the connection between the Simulator Exchange and Simulators, and between the Experimental frame Exchange and the Experimental frames

This architecture is built upon the OSM framework described in Chapter II. Winfrey, Baldwin, Cummings, and Ghosh (2014) tells us that OSM allows the “plug and play” of different components which “can be developed separately as plug-ins and combined to form a complete M&S system” . An OSM-based simulation consists of a collection of “input plug-ins (model, Experimental frame), execution plug-ins (Simulator), output plug-ins, the OSM executable, and the OSM library” (Winfrey, Baldwin, Cummings, & Ghosh, 2014). This framework is shown in Figure 13 with our Experimental frame Exchange, Simulator Exchange, and Metrics Collector added in.

Figure 13. OSM Software Architecture with the Metrics Collector, Experimental frame Exchange and the Simulator Exchange



In order to provide a way to identify and quantify emergent behavior, we have created a Metrics Collector, which interfaces with the Simulator Exchange, to collect metrics data from the agents and the Simulator Exchange. We define emergent behavior to be that behavior that comes about as one component that is affected by another component. In order to do this, the elements of the simulation shown in Figure 13 (regardless of the models) must not be unique to a simulation; they must be of use to any simulation without change to any of these elements. We also must be able to provide elements that can accommodate any needs of the simulation and the end user. In order to achieve both of these we must be able to have multiple Simulators and Experimental frames that are reusable and swappable across and within simulations.

The Simulator Exchange and the Experimental frame Exchange are designed to require only one Simulator to drive multiple simulations and one Experimental frame that allows the user to make parameter changes to the models over many configurations for one simulation. In addition to this reusability, we want to make these Simulators and Experimental frames so that the models in one simulation do not have to change when the Simulator and/or the objectives of the simulation change. We want the Simulator and/or the Experimental frame to be chosen at runtime by the user to allow multiple objectives in one execution of the SoS M&S.

This design provides a Simulator Exchange that allows one Simulator per simulation method (i.e., DEVS, DTSS, and DESS) to be used by any SoS M&S in order to provide maximum reusability for all SoS simulations. In addition, this design provides an interface to this Simulator such that Simulators are swappable within a simulation. For example, this means that the end user of a simulation can switch from a discrete event (DEVS) simulation to a discrete time (DTSS) simulation without changing any of the other elements (i.e., the models). In addition to the known simulator types (DEVS, DTSS, and DESS), a programmer can create other types, such as human in the loop, or real time, that have not yet been identified.

Because we can do this same exchange method with the Experimental frames, this design provides an Experimental frame Exchange that allows the reuse of an Experimental frame to be used by all SoS M&S. This methodology also allows these

Experimental frames to be swappable within one simulation. We call this set of Experimental frames the Experimental frame Exchange. This swappable Experimental frame Exchange allows any objective to be chosen at runtime for an SoS M&S without changing any of the other elements in the simulation.

As part of the architectural design for these Exchanges and Metrics Collector, we have developed the class diagram and collaboration diagram for these elements together. Figure 14 shows the class diagram for these elements and how they relate to each other. Figure 15 shows the collaboration diagram of how these elements interact.

Figure 14. Simulator Exchange/Experimental Frame Exchange/Metrics Collector Class Diagram

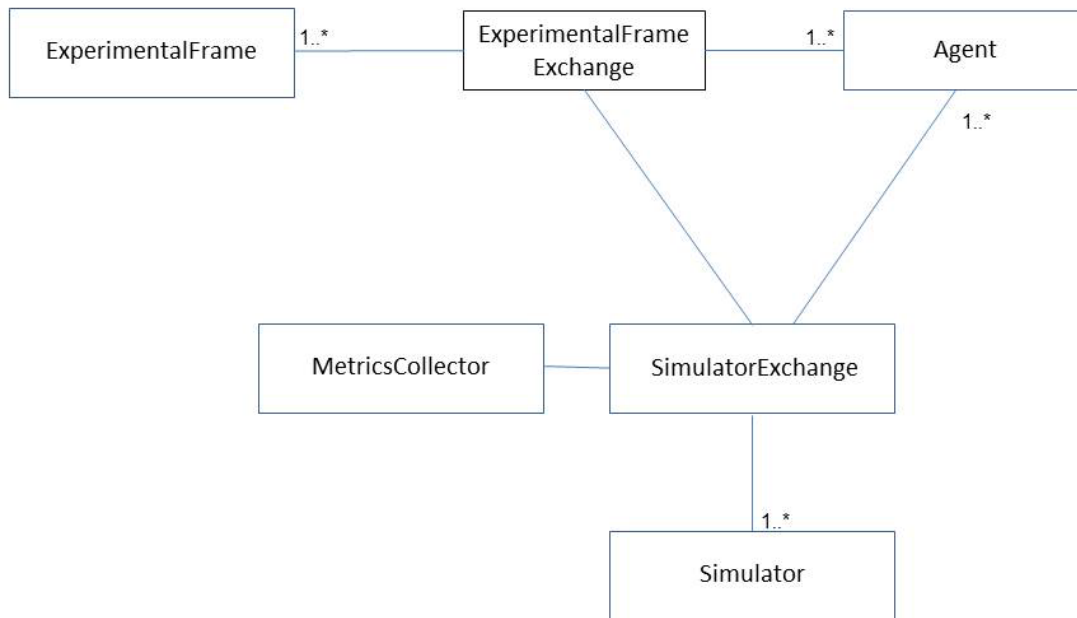
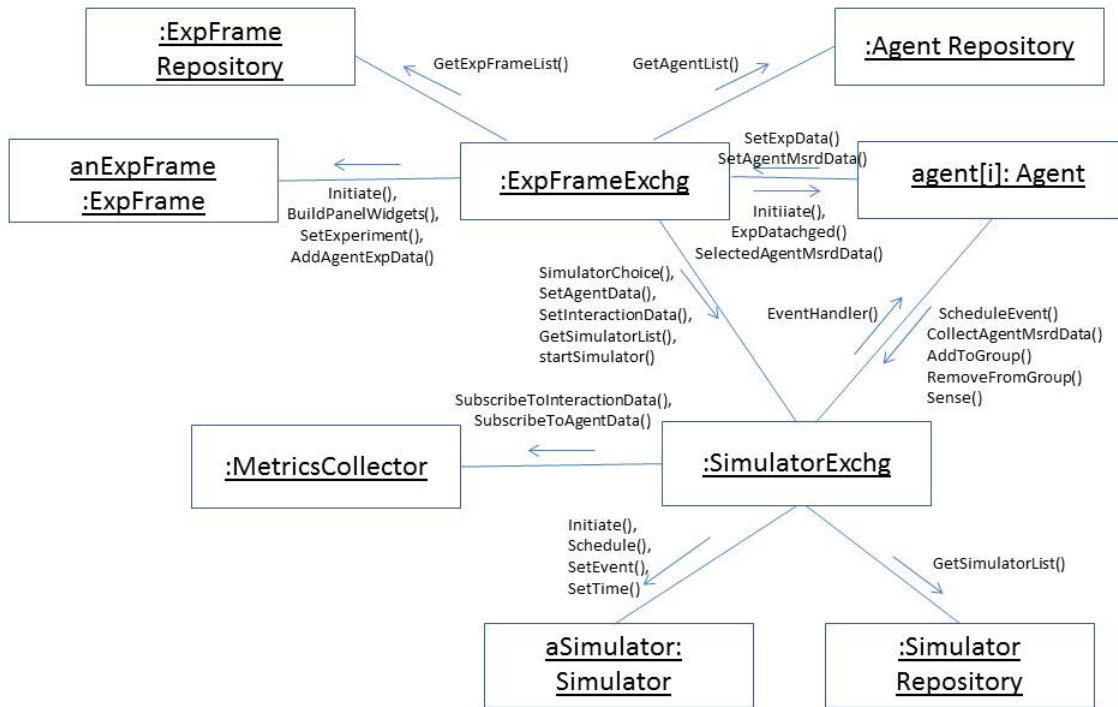


Figure 15. Simulator Exchange/Experimental Frame Exchange/Metrics Collector Collaboration Diagram



A. AGENTS

(1) Intent

Each agent in a SoS simulation represents some element of the real or source system; it is a model of that element. Using the OSM framework with the Metrics Collector, Simulator Exchange, and Experimental frame Exchange, the agents determine what experiment data and what measured data is to be used/collected. This measured data is then sent via the Simulator Exchange to the Metrics Collector during the simulation run. The Experimental frame Exchange and the Simulator Exchange provide standardized interfaces to be used by the agents regardless of the type of Simulator and Experimental frame the user has selected to use for a run of the simulation.

In order to develop these agents, the developer must first determine how to represent these agents. This work has found that using Finite State Automata and Finite

State-Time Automata, as discussed in the Swappable Simulators sections later in this chapter, works well for describing the states of the agent and the transitions that force these state changes.

(2) Motivation

The motivation behind the agents is to model some element of a System of Systems. Each agent can be as atomic as necessary.

(3) Applicability

Agents are the models that are executed through time in the simulation. They are a necessary part of a simulation.

(4) Consequences

- Agents must use the standard interfaces as specified by the Simulator Exchange and the Experimental frame Exchange. One such interface is the event handler. Each agent is required to have an EventHandler method. This is called by a Simulator. The number of times called and when are driven by what Simulator the user selected.
- Agents are expected to interact with the Simulator and other agents via events. In order to know what agent to interact with, an agent senses its environment through the Simulator Exchange.
- In order for agents to interact without causing changes to other agents when an agent is added, removed, or changed, standard interfaces and interface layers must be specified.

(5) Implementation

At initialization, an agent (through its constructor) sets its experiment data and its data to be measured. These are done through the agent's invocation of the Experimental frame Exchange's SetExpData, and SetAgentMsrdData methods.

In order to create an SoS simulation, the agents must interact without causing the other agents to change when an agent is added, removed, or changed. This is done through layers of interfaces. We use Java interface classes and abstract classes to do this.

Let's look at Predator Prey as an example. For this example, we will start with three agents: Grass, Rabbits, and Wolves. Rabbits eat grass, and wolves eat rabbits. All

agents in Predator Prey die. We will create an interface class called `LivingThing`, and it will have two abstract methods – `Die()` and `Eaten()`. These methods are called from the agent's `EventHandler()`, which is called from the `SimulatorExchange()`, as shown in Figure 15. All agents in our current Predator Prey simulation will implement the `LivingThing` class. Our agents are either plants or animals so we will create two implemented classes—`Plant` and `Animal`. The `Grass` will extend from `Plant`, and the `Rabbits` and `Wolves` will extend from `Animal`. These two classes will place themselves in groups that are visible to other agents. When a `Rabbit` wants to find the closest plant to eat, it will search in the group of `Plants` for the closest one. This is the same for `Wolves`. We can now add another agent. Let's add a `Weed`. This is also a `Plant`, but if this is eaten, it will kill the `Rabbit` when eaten. It can do this by overriding the `Eaten()` method in the `Plant` class to kill the agent that just ate him. The class diagram for these agents is shown in Figure 16. Instead of a `Weed`, let's add a `Tractor`, a different kind of agent that we did not consider before. In this case, we would create another interface class called `Moving Object`. It will have abstract methods for `Move()`, `Stall()`, etc and `Cover()`. Because we expect to add to this in the future, we will also create another implemented class – `Vehicle`. Any vehicle that covers a `Plant` will cause that plant to die while it is covering it. We will then create the `Tractor` class to extend from this vehicle class. We did not have to change the grass or weeds to add this new agent. Like the `Die()` and `Eaten()` methods of the `LivingThing` class, the methods of the `Vehicle` class are called from the agent's `EventHandler()` method. The class diagram for this new agent is shown in Figure 17. Note that the `Living Things Agent` and `Moving Objects Agent`, shown in figures 16 and 17, are examples of the `Agent1` subclass shown in Figure 19.

Figure 16. Predator Prey Example Class Diagram

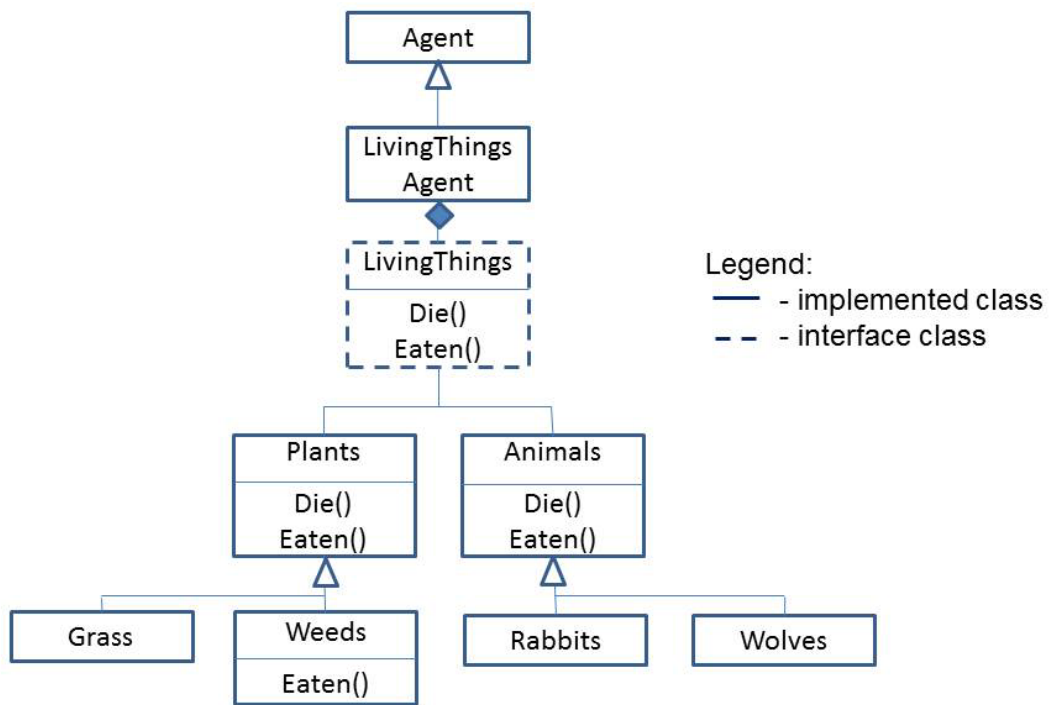
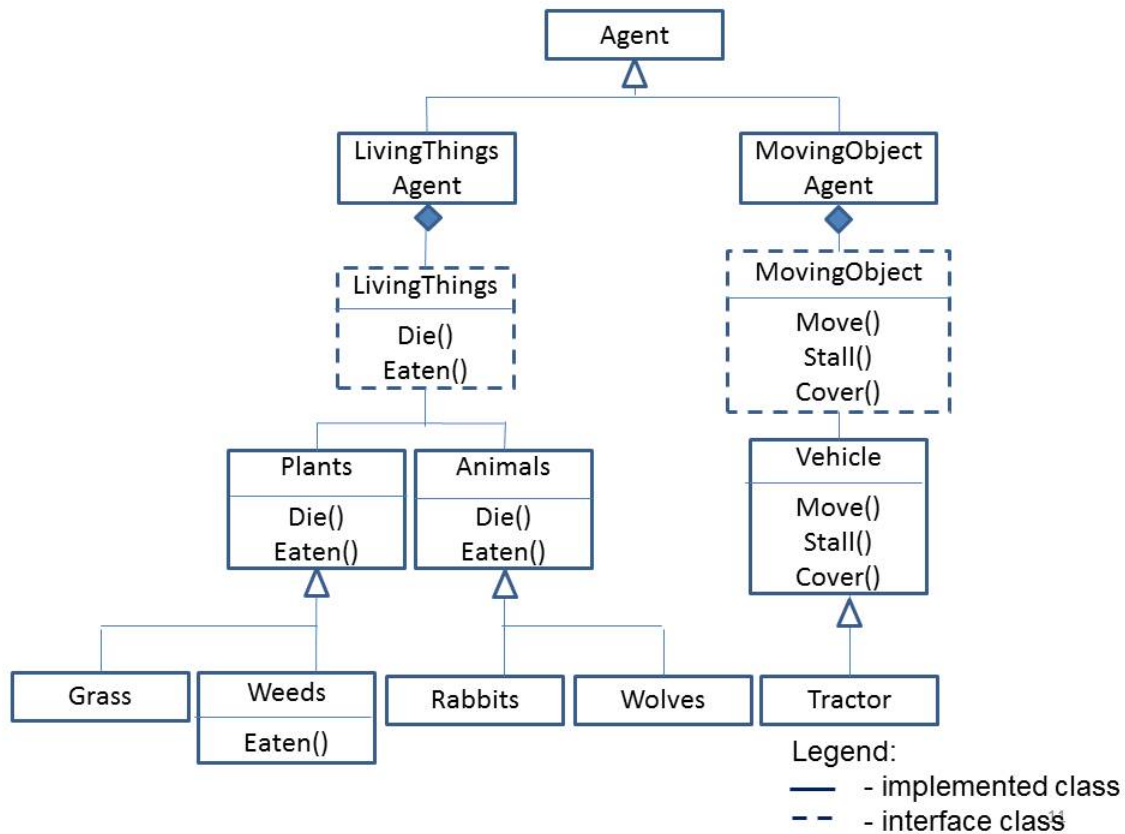


Figure 17. Predator Prey Example Class Diagram with Addition of Tractor Agent



Can we do this with other agents/simulations? Yes. Let's look at a surface warfare simulation that is defending a ship against a small boat threat. We can have four agents: ship, gun, sensor, small boat threat. We can define three classes these agents extend from: Target, Identifier, and Receiver. We can also define interface classes for these. The sensor is an Identifier for identifying the threat. It has a certain detection range. The ship is a receiver of messages from the Identifiers that the small boat threat (target) as hostile. Once we have run this simulation, let's add a new agent—Unmanned Aerial Vehicle (UAV) that will now do the identification of the small boats more efficiently. It will extend from the Identifier class, and will override methods to have a longer range than the sensor because of its ability to fly. None of the other agents had to change because of the addition of the UAV agent.

An agent implements its Finite State Automaton/Finite State-Time Automaton. As part of its initialization, an agent sets its initial state and the metrics it wants to create. It sets the tasks for each state through its private StateHandler method. Tasks may include transitioning to another state (using its private StateTransitionHandler) or scheduling events (like destroying another agent). Both of these tasks are handled by calling the Simulator Exchange's ScheduleEvent method. For example, to transition to another state, the agent would make a call to the Simulator Exchange's ScheduleEvent method with an event containing the sending agent and "TransitionState" as input. To destroy another agent, the agent would be a call to the Simulator Exchange's ScheduleEvent method with an event containing the target agent and the "destroy" as inputs. An agent is aware of other agents around it by invoking the Simulator Exchange's Sense method. For example, a rabbit can sense if any grass is close to it by calling this method to look for any agents in the plant class (from above) around it.

The Simulator Exchange's SetEvent method puts the event into its event queue, and causes the Simulator Exchange to inspect its event queue for the next event to be executed. It then calls the affected agent's EventHandler method. An agent makes the decisions to transition to another state through its private StateTransitionHandler method. See Appendix A for pseudo code for predator prey agents. It causes the event queue for the next event to be executed.

Figures 18 and 19 show the agent class diagrams which describe the classes needed in an agent for interactions with the Experimental frame Exchange and Simulator Exchange. Agent1 refers to any agent in simulation. The Agent class is the parent of any agent.

Figure 18. Agent Class Diagram

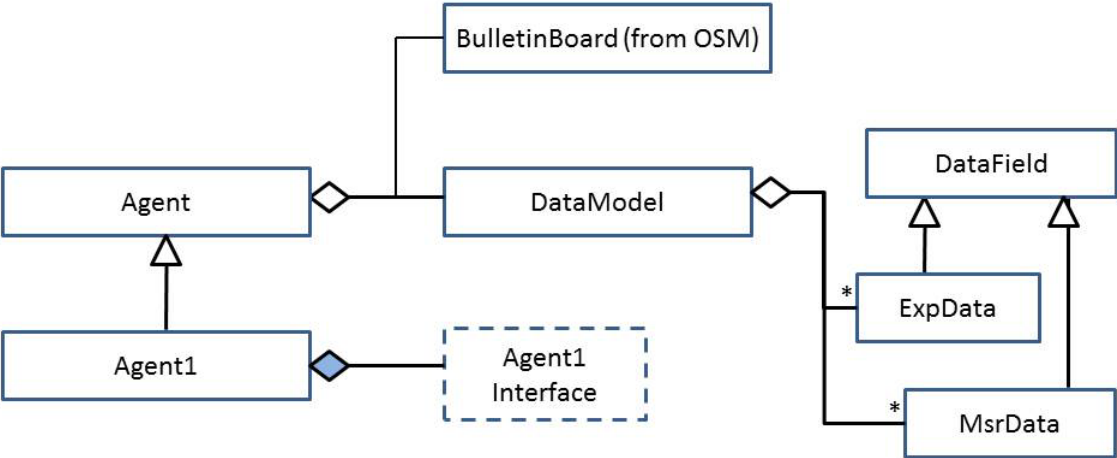
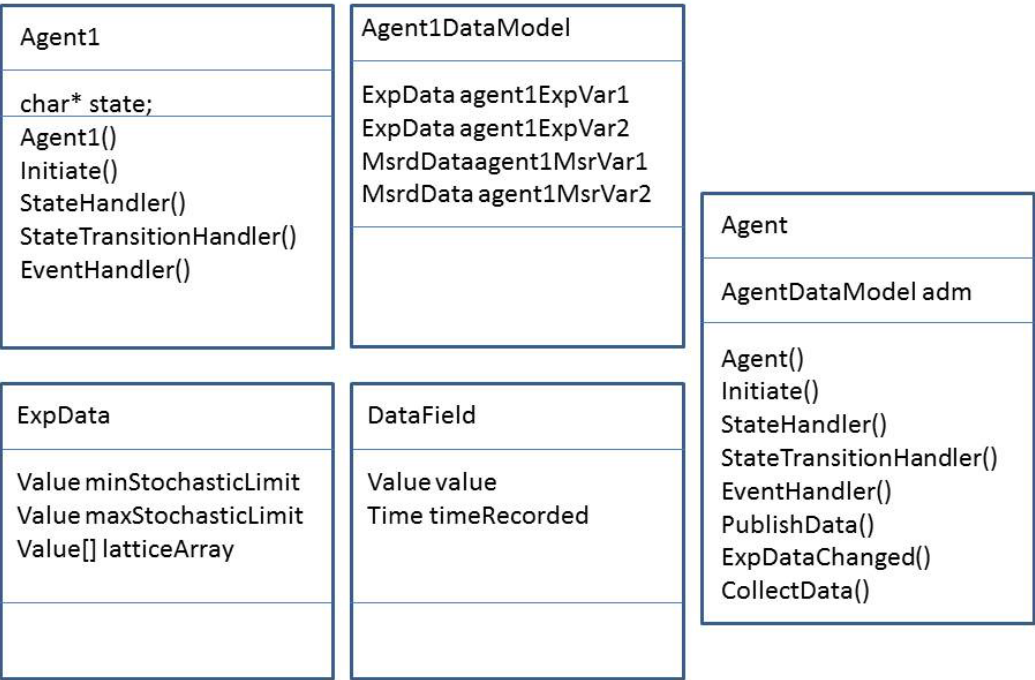


Figure 19. Agent Class Diagram




B. SWAPPABLE EXPERIMENTAL FRAMES

Through this research, we have designed swappable Experimental frames by developing three Experimental frames that can be used in multiple simulations and can be swapped in the same simulation. These frames are used through the Experimental frame Exchange. In all the figures, notice that the selection of metrics is in all of the types of Experimental frames. These Experimental frames are:

- Multi Run Experimental frame which allows the setting of input parameters without any randomization. See Figure 20 for an example of this.

Figure 20. Example of a Multi Run Experimental Frame Input Panel



The screenshot shows a window titled 'Experiment' with a close button (X) in the top right corner. The window contains several input fields and checkboxes arranged in a grid-like fashion. At the top left, there is a 'Time Step' field with the value '1.0'. Below it is a 'Simulation' dropdown menu set to 'DEVS'. Further down is an 'Initial Seed' field with the value '1528729'. To the right of these fields are several checkboxes: 'Rabbit effective' (checked), 'Wolf born' (unchecked), 'Rabbit energy loss' (unchecked), 'Wolf effective' (checked), 'Rabbit left' (unchecked), 'Wolf energy loss' (unchecked), 'Grass / 4' (unchecked), 'Rabbit regular' (checked), 'Wolf left' (unchecked), 'Grass effective' (checked), 'Weeds / 4' (unchecked), 'Wolf regular' (checked), 'Grass regular' (checked), 'Weeds effective' (unchecked), 'effective' (checked), 'Rabbit born' (unchecked), 'Weeds regular' (unchecked), and 'regular' (checked). At the bottom right of the window is a 'Run' button.

- Lattice Experimental frame which provides the setting of parameters (selectable) to run through a set of numbers. See Figure 21 for an example of this, showing two of the numbers.
 - Set number of runs based on number of values in input set
 - Ex: variable = Lower limit 1, upper limit 4—runs 4 times with variable =1, variable = 2, ...

Figure 21. Example of a Lattice Experimental Frame Input Panel

- Stochastic which allows the setting of 1 parameter to pick a random number between a lower and upper limit for a set number of runs. See Figure 22 for an example of this.
- Ex: variable = 5, + or -2-10 runs; variable is set to some random number between 3 and 7 for 10 runs

Figure 22. Example of a Stochastic Experimental Frame Input Panel

(1) Intent

The intent of this work is to provide a design for developing an Experimental frame Exchange that provides Experimental frames (as called out in Zeigler's (1976) book) that are swappable. Let's first define what an Experimental frame is. It defines the objectives of the simulation in terms of a set of parameters. For a SoS simulation, it must define these objectives such that they encompass all of the component systems.

The term "swappable" means to trade or exchange something for another. In the case of an Experimental frame, this means to be able to trade or exchange the objectives of a simulation without changing any of the other elements of the simulation. This means that the Simulator and the models do not change as a result of swapping or exchanging one Experimental frame for another.

(2) Motivation

As we described in Chapter I, many M&S are being written for users where every time the objectives of the simulation change, the simulation has to be rewritten in some areas or the whole simulation has to be rewritten. We believe that simulation objectives can be defined in such a way that the models and the Simulator do not have to change. In addition, when multiple simulations are written for the same set of objectives (in terms of the same set of parameters) by an organization, they should not have to recreate an Experimental frame for each simulation.

This methodology should be used whenever more than one simulation is written by an organization. It should also be used whenever a simulation may have multiple objectives.

(3) Applicability

Use the Experimental frame Exchange when

- Multiple simulations will be written by an organization
- One simulation will be written that can have multiple objectives (using one objective at a time)

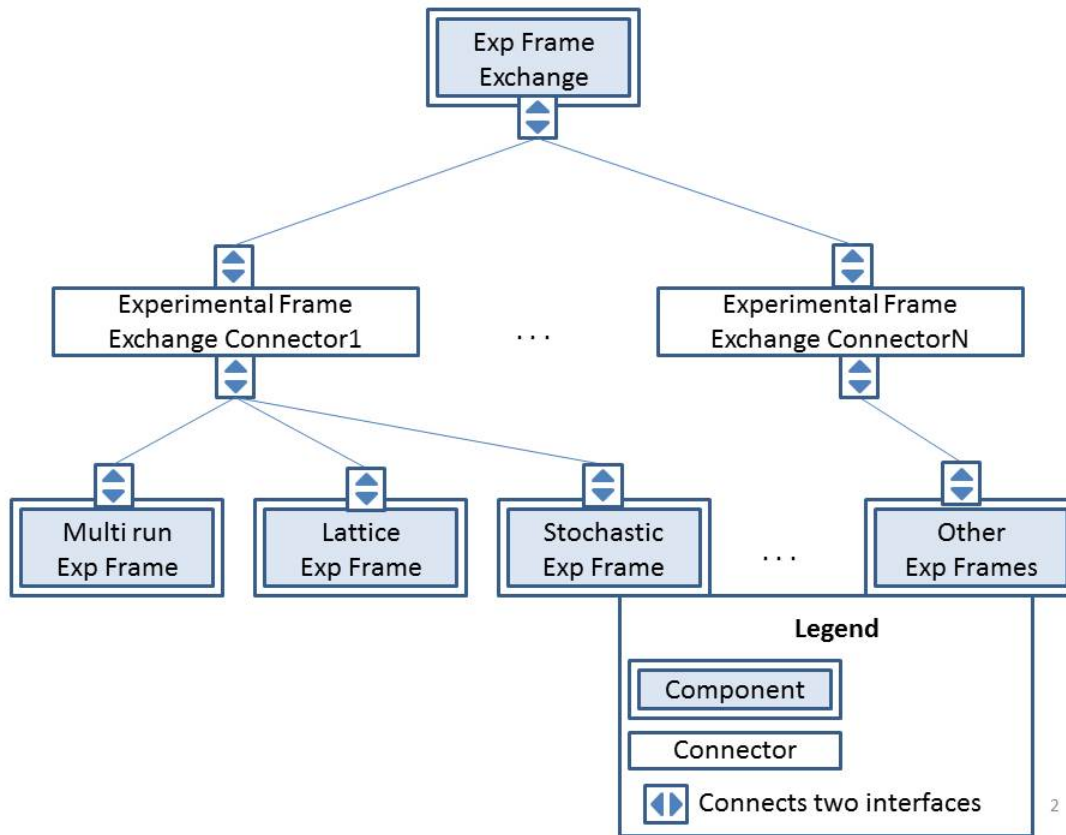
(4) Consequences

- In order for the Experimental frame Exchange to be swappable, it must define the inputs and outputs of it. Each model must provide inputs to be shown in the parameter input panel when created through the Experimental frame Exchange.
- Because the Experimental frame Exchange provides the panels that display the parameters, it may not have the optimum “look” for the panel. In other words, there may be extra space in the panel that would not be there if it was built by the developer manually.

(5) Implementation

The Experimental frame Exchange uses the C2 architecture style to achieve its goals as shown in Figure 23. The C2 style is used to allow an Experimental frame to be chosen from any number of Experimental frames without causing changes to the Experimental frame Exchange, the Simulator Exchange, the Metrics Collector, or the models.

Figure 23. Experimental frame Exchange and Experimental frames Connected Via the C2 Architectural Style

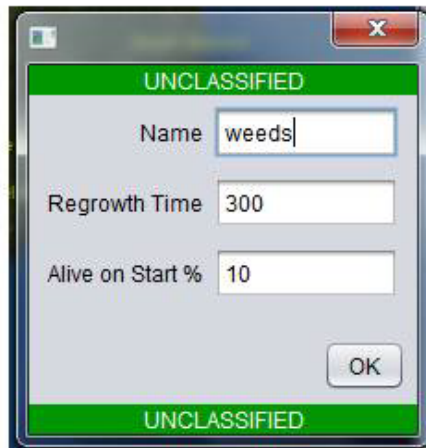


The Experimental frame Exchange performs the following actions:

- The Experimental frame Exchange provides lists of the following for the user to choose from.
- Available Experimental frames
- Available Simulators
- Available agents
- By retrieving all the agents in the agent repository, it provides a panel of the agent names for the user to choose from. It then loads all the chosen agents to create the SoS simulation, in the order they were selected.
- It provides a panel of input parameters to the user for each agent chosen. Once values are set, this data is now known as the experiment data. See Figure 24 for an example of this type of panel.

- It passes the experiment data to the agents and to the Metrics Collector.

Figure 24. Example Input Parameter Panel



The image shows a standard Windows-style dialog box. The title bar is blue with a green header area containing the word 'UNCLASSIFIED' in white. The main area has a light gray background. It contains three labeled text boxes: 'Name' (containing 'weeds'), 'Regrowth Time' (containing '300'), and 'Alive on Start %' (containing '10'). At the bottom right is an 'OK' button. The footer area is green with the word 'UNCLASSIFIED' in white.

To show the implementation of the Experimental frame Exchange, we show the class diagrams, the use case for using the Experimental frame Exchange, and the sequence diagram based on that use case. These are seen in Figures 25, 26, and 27.

Figure 25. Experimental Frame Exchange Class Diagrams

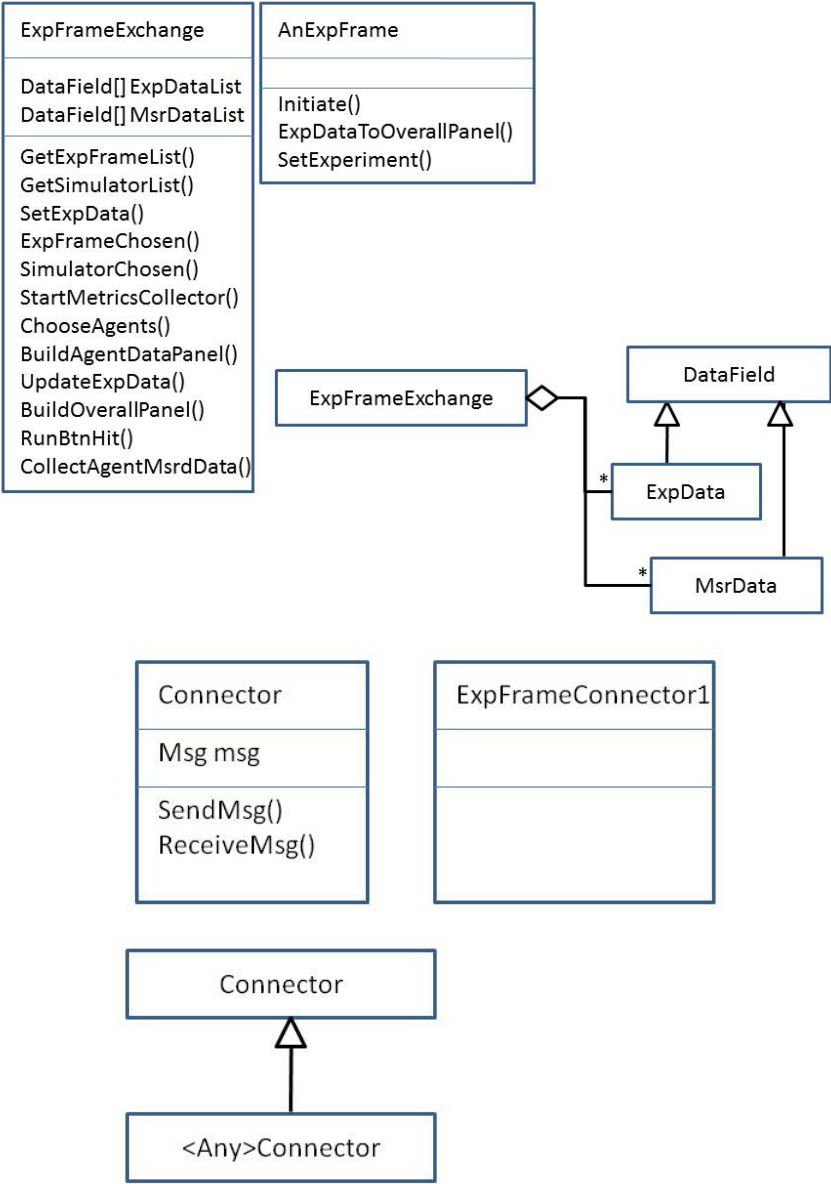
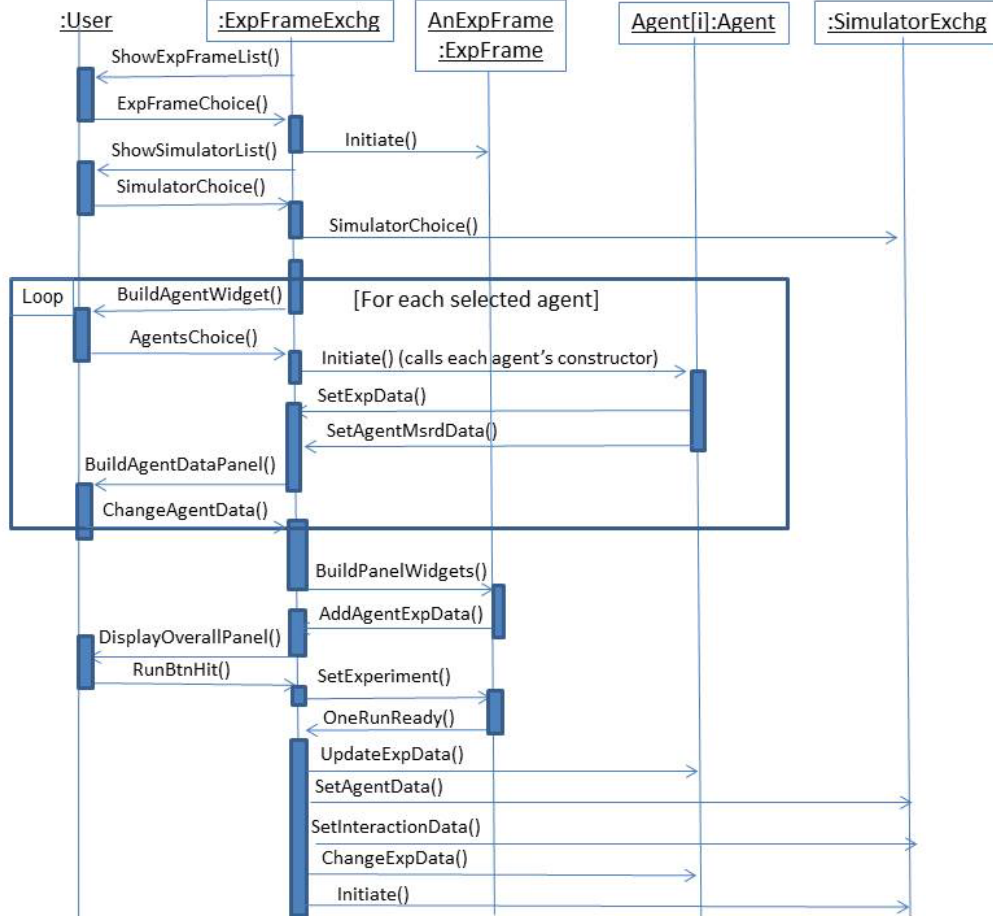


Figure 26. Experimental Frame Exchange Use Case Description

Use case: UC-1. Experimental Frame Objectives Setting
Primary Actor: Experimental Frame Exchange
Preconditions: User starts the simulation
Postconditions: Simulation has ended
Main Scenario:

1. Call GetExpFrameList() to provide list of Experimental Frames to the user
2. User selects the Experimental Frame
3. Initiate chosen Experimental Frame
4. Call GetSimulatorList() to provide list of simulators to the user
5. User selects the Simulator
6. Pass Simulator choice to Simulator Exchange
7. Call GetAgentList() to provide list of agents to the user
8. User selects an agent
9. Initiate chosen agent
10. Get Experiment data from the agent, user updates experiment data
11. *Repeat steps 8-10 until agent selection is done*
12. Build overall experiment panel
13. Display overall experiment panel
16. Add agent data to overall panel dependent on Experimental Frame
17. User hits run button
18. Set up one experiment
19. Set Agent Data for simulator
20. Set interaction data for simulator
21. Change experiment data
22. Start simulator for one run

Figure 27. Experimental Frame Exchange Sequence Diagram



C. SWAPPABLE SIMULATORS

Through this research, we have designed swappable Simulators by developing three Simulators that can be used in multiple simulations. These Simulators can be swapped through the Simulator Exchange. We first developed the Discrete Simulator as a baseline for this work. The Discrete Simulator can drive both discrete events and discrete time simulations. Both the DEVS and DTSS Simulators are connected to the Simulator Exchange through the C2 architecture. Because of this, these Simulators are swappable. Thus, the models (agents) will not change when swapping between each of these Simulators. Therefore, the user can select (through the Experimental frame Exchange) the simulation type, and get possibly different results without changing the rest of the simulation modules.

(1) Intent

The intent of this Exchange is to provide a design for developing a Simulator, as called out in Zeigler's book, which is swappable. As stated earlier, a Simulator executes a model through time. A swappable Simulator uses a different simulation algorithm (or simulation type) without changing any of the other elements of the simulation. This means that the Experimental frame and the models do not change as a result of swapping or exchanging one Simulator for another. This Exchange uses the C2 architectural style to allow the use of both Simulators (DEVS and DTSS) without changing the models (agents).

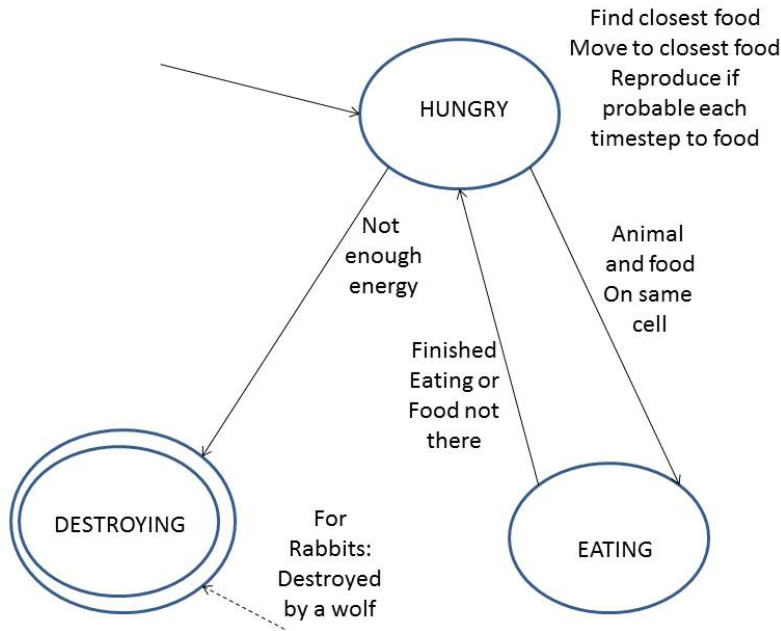
(2) Motivation

As we described in Chapter I, many M&S are being written for users where the Simulator is either embedded in the model or the Simulator is written to drive one model or one class of models. We believe that Simulators for a particular simulation algorithm or type can be written such that they can drive all models and they will not require changes to the models. This means that the models can be used without change for either Discrete Events, Discrete Time, or Differential Equations.

The motivation behind the Simulator Exchange is the use of the Components and Connectors (C2) Architectural Style to use the same simulator engine differently based on the simulation type chosen. If the Discrete Event simulator type is chosen, an agent only moves to the next state if an event has occurred. Events can be caused outside the agent's control. These are called external events. The agent can also schedule events for himself—these are called internal events. These events can be caused by time or by some occurrence, such as an agent landing on the same cell as another agent. We use a Finite State Automaton to show the transition through the states, as shown in Figure 28. The phrases next to the lines are the inputs that cause the state transition.

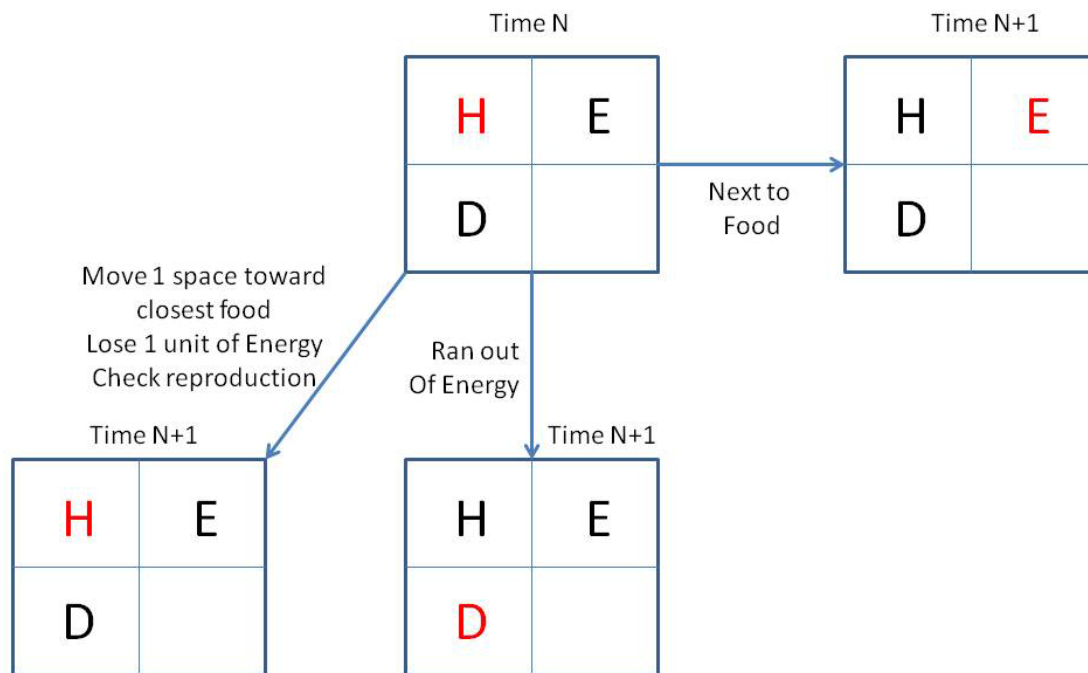
Figure 28. Finite State Automaton

Wolf/Rabbit



If the Discrete Time simulator type is chosen, each agent is checked for a possible change in state at each time step, as in Figure 29. According to Zeigler (2000), DTSS is a stepwise model of execution. We use states as events as specified in the Agent section earlier in this chapter. At any timestep, “the model is in a state and the formalism defines what the next state will be at the next timestep.” “The next state depends on the current state and the environment’s influences.” We are using the environment’s influences to be the inputs at that particular timestep. We are using a variation of a Finite State Automaton, called a Finite State-Time Automaton (FS-TA) to show the progression of states for each timestep in a DTSS. Figure 29 shows an example of a FS-TA. According to Zeigler (2000), all agents undergo a state transition at each timestep. This occurs whether or not an agent undergoes a transition. Once all state transitions have been computed, the clock advances to the next time step.

Figure 29. Finite State-Time Automaton



(3) Applicability

Use the Simulator Exchange when

- Multiple simulations of various simulation types will be written by an organization
- One simulation will be written that changes its performance based on which simulator type is chosen

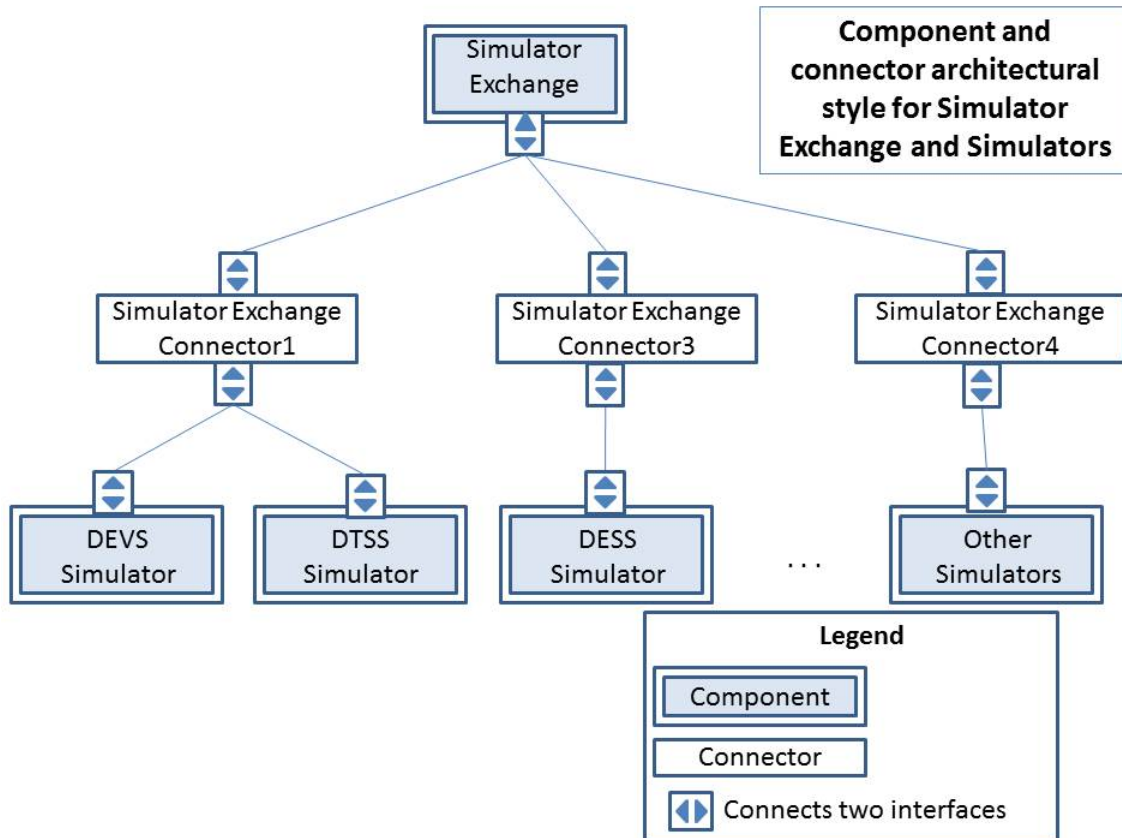
(4) Consequences

- Each model must use predefined methods for state transition and states. These are the Simulator Exchange's Event method and agent's EventHandler method (which is called by the Simulator).
- The simulator uses C2 architectural style.
- Simultaneous events will be handled as they are scheduled. In other words, if two events are scheduled for the same time, the first one scheduled will be executed. This can cause unexpected effects.
- Each agent must be written using the specified format. This allows the switching between DEVS and DTSS without changing the code.

(5) Implementation

The Simulator Exchange uses the C2 architecture style to achieve its goals as shown in Figure 30. The C2 style is used to allow a Simulator to be chosen to from any number of Simulators without causing changes to the Simulator Exchange, Experimental frame Exchange, the Metrics Collector, or the models. In this figure we see that there is one connector between the DEVS and DTSS Simulators, and there is a connector-Simulator pairing with the other Simulators. The choice of whether to have one Simulator or more than one attached to a connector is made when the Simulator has been designed. In the case of the DEVS and DTSS Simulators, we can see enough similarities in the two algorithms that it works best to have one connector.

Figure 30. Swappable Simulator Architectural Style



The Simulator Exchange performs the following actions:

- At the start of the simulation, the Simulator Exchange executes the initiate method of each agent in the order that each agent was selected in the Experimental frame Exchange.
- Each state transition (for each agent) is scheduled as an event (with the name “targetState”). For this event, the EventHandler method calls the agent’s StateTransition method. Each state executes some set of tasks. These tasks may include scheduling an event or transitioning to another state. Both of these are done in the agent’s StateHandler method.
- If the DEVS simulation type was chosen:
 - When the event or timestep has occurred, the Simulator Exchange executes the EventHandler method. This method will determine if the next event should be executed. This could be the switching to the next state.
- If the DTSS simulation type was chosen:
 - The Simulator Exchange executes the EventHandler method for every second until the desired timestep is achieved.
- At the end of each event, the interaction metrics for the affected agents may be incremented.

The algorithms that are used for DEVS and DTSS are done through the connector. The simulator exchange does not know what Simulator is connected. The simulator exchange’s only interaction with the specific simulator is when it initiates it. Figures 31, 32 and 33 show the class diagrams, the use case for the Simulator Exchange, and the sequence diagram for that use case.

Figure 31. Simulator Exchange Class Diagrams

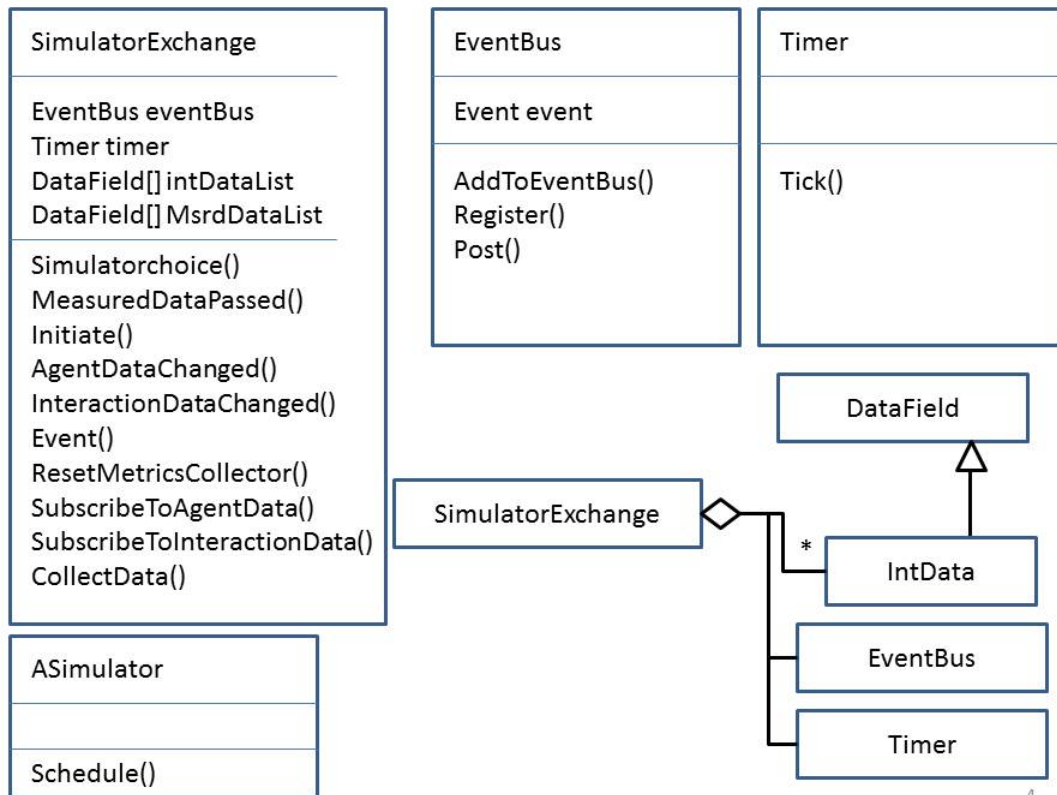
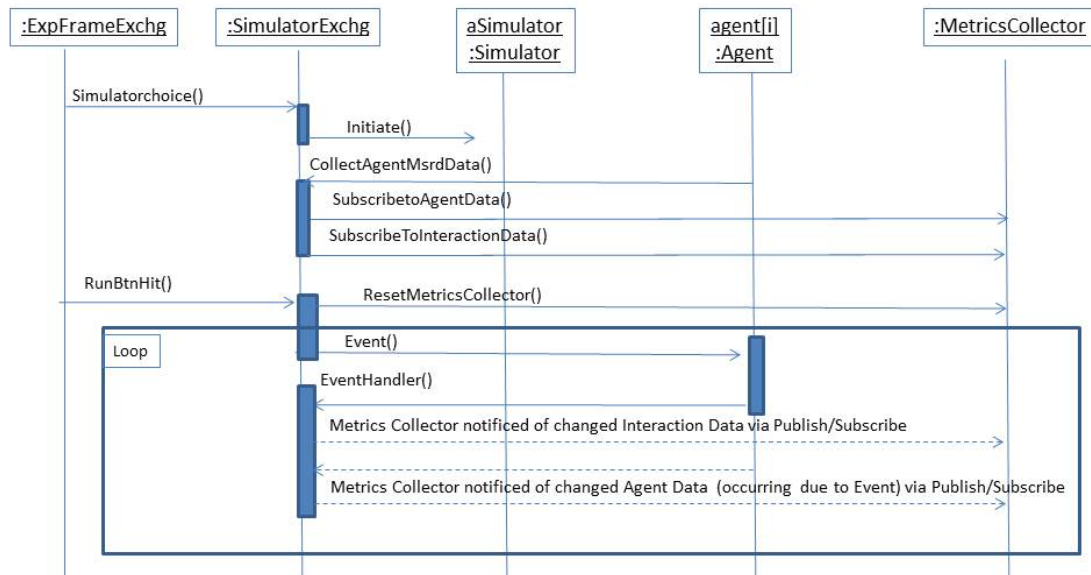


Figure 32. Simulator Exchange Use Case Diagram

Use case: UC-2. One Simulation Run
 Primary Actor: Simulator Exchange
 Preconditions: Simulator choice is passed to the SimulatorExchange
 Preconditions: User selects measured data (both agent data and interactions data)
 Postconditions: Simulation has ended
 Main Scenario:
 1. Initiate chosen Simulator
 2. Agent provides measured data
 3. User selects the Run button
 4. Simulator Exchange resets the Metrics Collector
 5. Metrics Collector subscribes to the data the Simulator Exchange is holding
 6. Agent executes the statehandler for the initial state
 7. Agent submits events (one being transition state event)
 8. Simulator calls the event handler for affected agent
 9. After event handler returns, notify all subscribers of changed interaction data
 10. When agent's measured data changes, notify all subscribers of changed agent data
Repeat steps 7 -10 for each agent

Figure 33. Simulator Exchange Sequence Diagram



D. METRICS COLLECTOR

Our research contends that we can collect all SoS metrics in one location where we can evaluate the data collected to determine if it can be identified as emergent behavior. This central collection point allows us to collect data across the System of Systems simulation. The metrics collected is selected by the end user. This data will be graphed for the end user on one graph.

This data is provided to the Metrics Collector using the Publish/Subscribe architectural style. The Metrics Collector subscribes to the agent metrics data published by the agent and to the interaction data published by the Simulator Exchange. This publishing and subscribing of data is done using OSM's Bulletin Board class. The Metrics Collector subscribes to this data through the Simulator Exchange's methods.

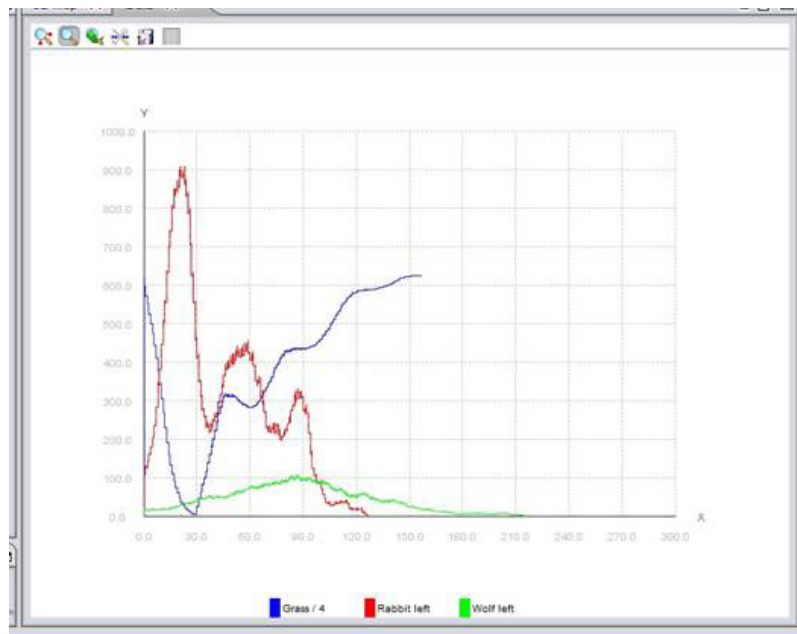
(1) Intent

The intent of the Metrics Collector is to collect metrics. There are two types of metrics data to collect:

- Agent data
- Interaction data

The agent data is defined by the agent, and is provided to the end user (through the Experimental frame Exchange) to select which of the agent data he/she wants to collect. An example of a graph of the set of agent data for the Predator Prey simulation is in Figure 34.

Figure 34. Predator Prey Agent Metrics

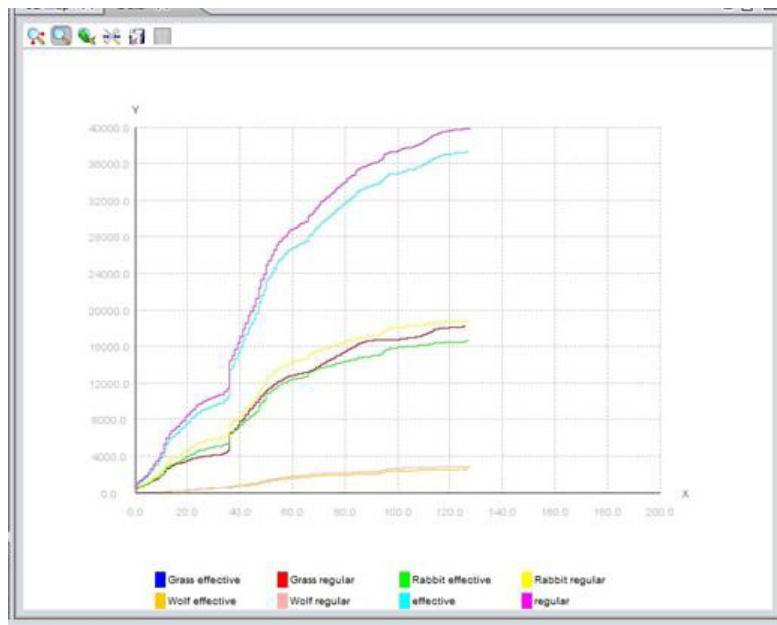


The interaction data is described in Chan (2011). This data is maintained and updated through the Simulator Exchange. In this article, a counter is increased whenever an agent has to interact with another agent based on some action that the agent has to perform. Chan graphs these interaction metrics and asserts that the interaction metric deviates from a normal curve when emergent behaviors arise. Per Chan, there are two types of interactions

- “Regular—agent initiates or receives a contact with another agent regardless of whether this action will finally induce any outcome”
- “Effective—regular interaction with a final outcome”

We can identify regular and effective interactions in the Predator Prey simulation. A regular interaction is when the rabbit moves to a cell where grass was, but the grass was already eaten by another rabbit. The rabbit then goes back into the hungry state. An effective interaction is when the rabbit moves to a cell with grass and eats it. An example of a graph of both effective and regular interaction metrics for the predator prey SoS simulation is shown in Figure 35.

Figure 35. Predator Prey Interaction Metrics



(2) Motivation

The motivation behind the Metrics Collector is to collect metrics from each of the agents and to collect interaction metrics from the Simulator Exchange to allow identification and quantification of emergent behavior. This data can be SoS metrics, such as performance of an SoS weapon system. We did this in such a way to not require various simulations to have to make specific or unique coding changes to set and collect these metrics.

(3) Applicability

Use the Metrics Collector when:

- A SoS simulation is built and to be used for the identification and quantification of emergent behavior.

(4) Consequences

- Each agent specifies the agent metrics that can be collected.
- The Simulator Exchange must provide a method for the Metrics Collector to subscribe to the agent data because it provides there is no connection between the agent and the Metrics Collector directly.

- The Simulator Exchange specifies the interaction metrics that can be collected.
- The values of the interaction metrics may be so much larger than the agent metrics that the agent metrics may not be as visible. In order to view the two sets of data together, the interaction metrics have been divided by 50 to make them appear on the graph in the same numerical area. Because we are looking changes in the data through time, we believe that doing this division will not impact the analysis.
- Whenever any of the data changes, the Simulator Exchange and the agent must notify all subscribers of this change by calling the Publish() method that is a part of OSM's Bulletin board class. Both do this through their own CollectData() methods.

(5) Implementation

The metrics collector subscribes to data published by the Simulator Exchange and the agent to achieve its goals. This is done through the Bulletin Board class provided by the OSM framework. The Metrics Collector does not interact with the agent directly. It calls the Simulator Exchange's SubscribeToAgentData() and SubscribeToInteractionData() to subscribe to the metrics data it needs to collect. To show the implementation of the Metrics Collector, we show the class diagrams, the use case for the use of the Metrics Collector and the sequence diagram that is built on this use case. These are seen in Figures 36, 37, 38, and 39.

Figure 36. Metrics Collector in the Publish/Subscribe Architectural Style through OSM's Bulletin Board

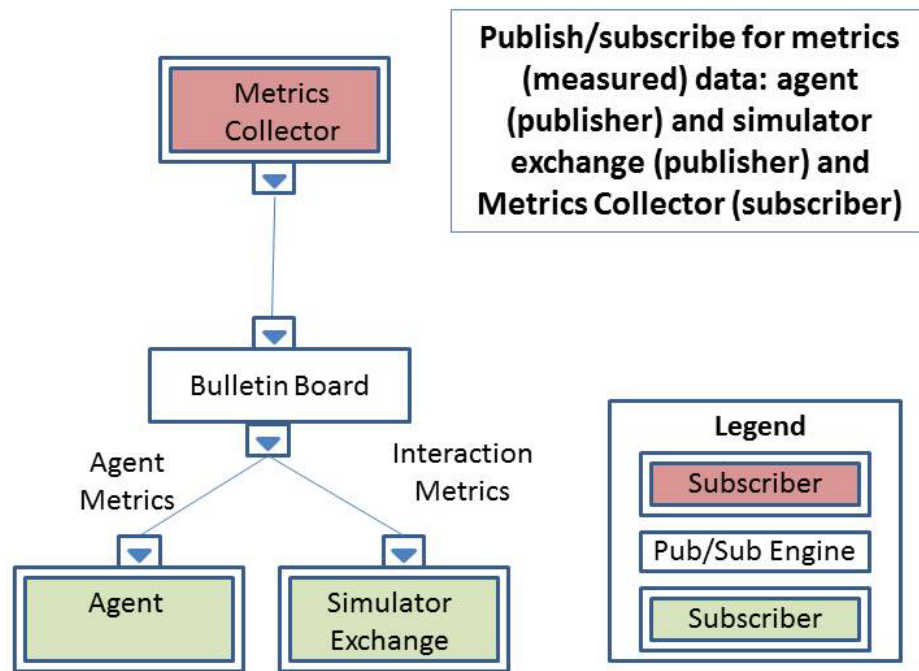


Figure 37. Metrics Collector Class Diagrams

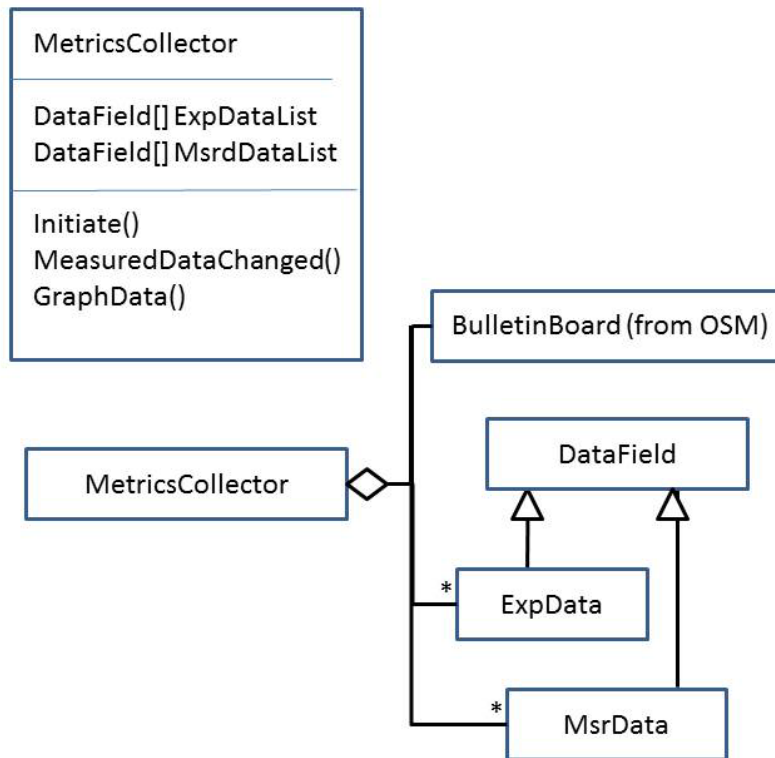


Figure 38. Metrics Collection Use Case

Use case: UC-3. Metrics Collection

Primary Actor: MetricsCollector

Preconditions: User has hit the Run button on the ExpFrameExchange Overall Panel

Postconditions: Simulation has ended

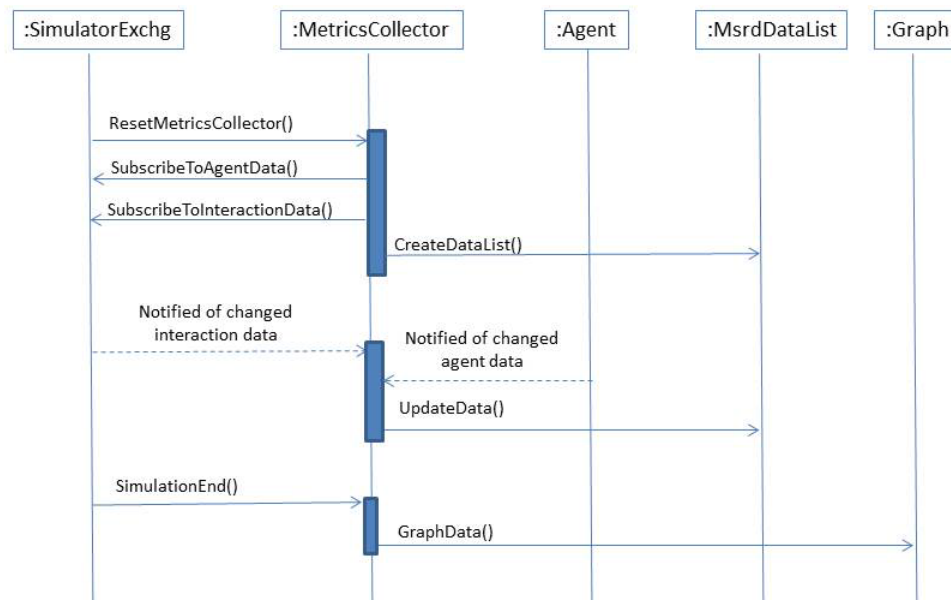
Main Scenario:

1. The Simulator Exchange resets the Metrics Collector
2. The Metrics Collector uses the SubscribeToAgentData() to subscribe to each agent's metrics data
3. The Metrics Collector uses the SubscribeToInteractionData() to subscribe to the interaction data
4. The Metrics Collector adds each piece of metrics data to the Measured Data List (MsrdDataList)
5. Whenever the Metrics Collector is notified of a change in any of this data, the Measured Data List is updated

Repeat step 5 whenever data is changed

6. When simulation has ended
 - a. Graph the agent metrics on the Graph
 - b. Graph the interaction metrics on the Graph
 - c. Display the Graph

Figure 39. Metrics Collector Sequence Diagram



THIS PAGE INTENTIONALLY LEFT BLANK

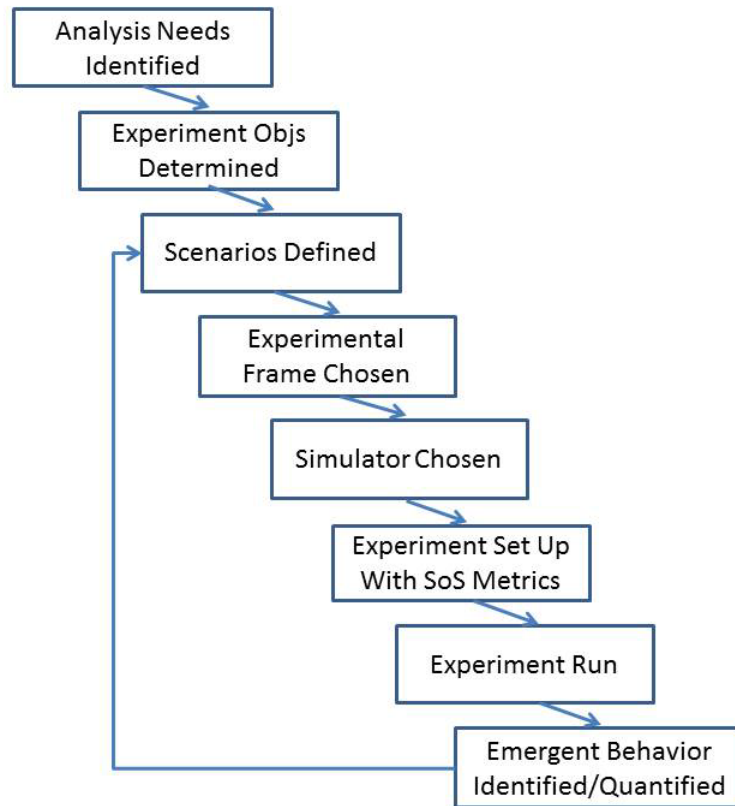
IV. EMERGENT BEHAVIOR IDENTIFICATION AND QUANTIFICATION

Validation is defined as the assurance that a product (e.g., a Weapon SoS) meets the needs of the customer and stakeholders. When it comes to an SoS, the Weapon SoS has to be run in all its configurations to validate the component systems and to validate that the interactions among the component systems are acceptable. As part of this validation, the behavior that emerges from the interactions of the component systems must also be known and determined to be acceptable or unacceptable. As stated in Chapter I, Modeling and Simulation (M&S) can be an affordable validation alternative to operationally testing all functionality and interfaces in a weapon SoS. With M&S we can identify and quantify the emergent behavior that occurs in all these different configurations, thus enabling the determination that this behavior is acceptable or unacceptable.

Using the software architecture defined in Chapter III, we can now define the process for this identification and quantification of emergent behavior. Figure 40 provides the diagram for this process. The following sections detail each step in this process.

This process assumes that the SoS simulation has been developed using this research. This means that the user will have access to all of the Experimental frames and Simulators discussed in Chapter III. The agent developers developed each agent making all of its inputs changeable by the user. This means that there should be no “hard wired” input values in the code of an agent. We will also assume that the agent developers have defined all of the possible agent metrics and interaction metrics. These interaction metrics define which events or state changes occur because of interactions between agents.

Figure 40. Emergent Behavior Identification and Quantification Process



A. ANALYSIS NEEDS IDENTIFIED

When a simulation is developed, it is done for some purpose. Some area needs to be studied. The user or organization will identify one or more analysis needs. It is through this analysis that it is determined if an agent based simulation can be used.

We are assuming that the agents have already been developed. It is then just a matter of “plug and play” for the user to set up a simulation that meets his analysis needs using the existing OSM framework and the components developed through this research.

B. EXPERIMENT OBJECTIVES DETERMINED

This step is to determine what the user wants from the simulation runs. Does the user need a discrete event simulation or does he need a discrete time simulation? Or is there another Simulator that needs to be developed to meet his/her needs? These same type of questions are asked for the other components of the simulation as well. Does the

right Experimental frame exist or do we need to create a new one? We must determine the type of metrics or output we want, and we then need to create new output plug ins if the existing ones do not suffice.

C. SCENARIOS DEFINED, EXPERIMENTAL FRAME AND SIMULATOR CHOSEN

The simulation user must first determine what scenarios to run. This is done by determining all the configurations of SoS that need to be tested. This involves determining which agents to run together.

Along with choosing the agents, the user must also determine what input values to use for each of the agents. It may be that this scenario will include some number of runs with different values for one or more of the experiment data. This will determine how many runs are needed for the scenario. This could be one run, multiple runs, runs in a lattice type scenario, or multiple runs with stochastic behavior for some set of input data. For the lattice type scenario (as described in Chapter III), some number of discrete values for one or more experiment data variables are needed. These decisions will determine the type of Experimental frame to use.

The user must also determine what metrics to collect. This includes the agent metrics along with the interaction metrics. In order to see emergent behavior, we contend that there needs to be both agent metrics and interaction metrics plotted on the graph together.

The user must also determine what simulation type is to be used. This research provides both DTSS and DEVS simulation types. Chapter VI describes other Simulators that can be built as future work.

D. EXPERIMENT SETUP AND RUN

After the scenario has been determined, the user can now run the simulation. To do this, the decisions made during scenario definition must be input to the simulation through the Experimental frame. See Figures 41 and 42 for examples of agent choice using the Predator Prey simulation. In Figure 41, the user chose to run the simulation

with rabbits, wolves, and grass. In Figure 42, the user chose to also add weeds to the simulation. This will make an entirely different run with different outcomes.

Figure 41. Agent Selection Example

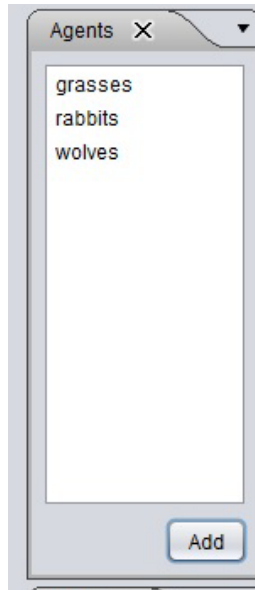
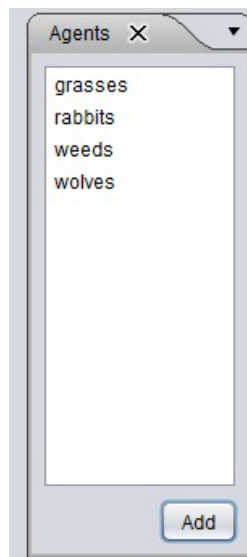
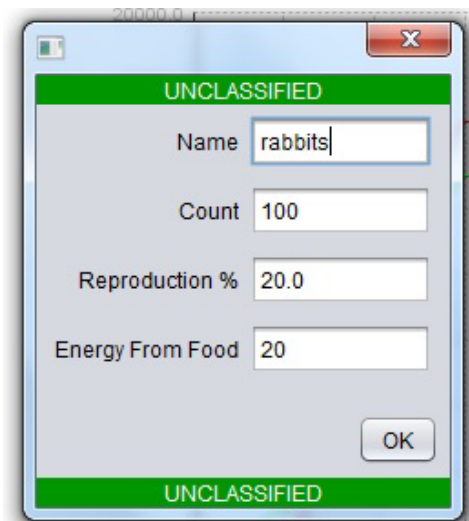


Figure 42. Another Agent Selection Example



As each agent class is selected in the overall Experimental frame GUI panel, the user is presented with the agent's available inputs. The values are then set according to the scenario defined. Figure 43 is an example input panel for an agent.

Figure 43. Example Input Panel



UNCLASSIFIED

Name rabbits

Count 100

Reproduction % 20.0

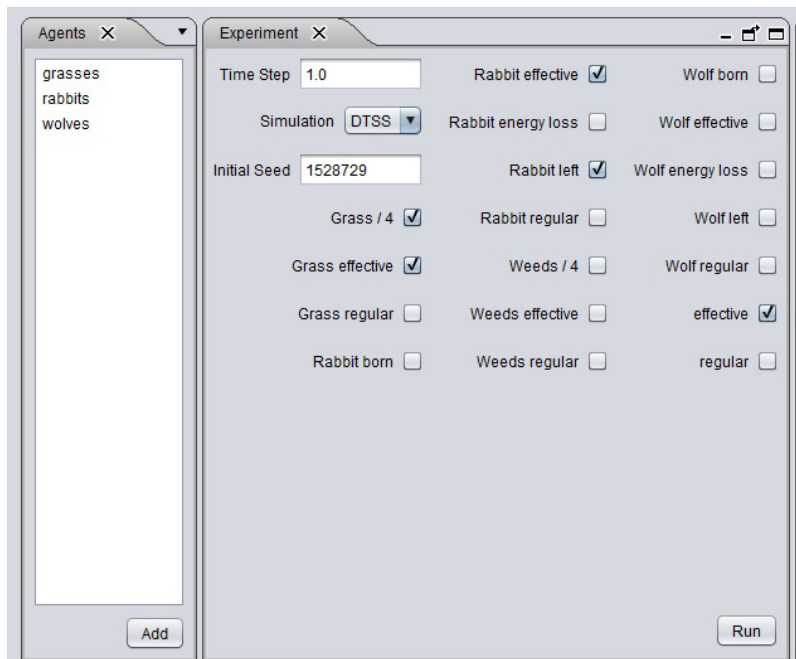
Energy From Food 20

OK

UNCLASSIFIED

The overall GUI panel for the experimental frame provides the user with all the metrics that can be collected in the simulation. Figure 44 shows an example of what metrics can be collected for the Predator Prey simulation.

Figure 44. Example Metrics to be Collected



Agents x Experiment x

grasses
rabbits
wolves

Add

Time Step 1.0

Simulation DTSS

Initial Seed 1528729

Grass / 4 ☒

Grass effective ☒

Grass regular ☐

Rabbit born ☐

Rabbit effective ☒

Rabbit energy loss ☐

Rabbit left ☒

Rabbit regular ☐

Weeds / 4 ☐

Weeds effective ☐

Weeds regular ☐

Wolf born ☐

Wolf effective ☐

Wolf energy loss ☐

Wolf left ☐

Wolf regular ☐

effective ☒

regular ☐

Run

As part of the scenario definition, the user determined what type of experiment is needed. Is it one run, multiple runs, multiple runs with stochastic behavior in one or more data values or a lattice type of experiment? The user chooses which type of Experimental frame to use based on those decisions. Figures 20, 21, and 22 in Chapter III show examples of these three types of run setups.

After the scenario has been input through the Experimental frame, it can be saved for future runs. The experiment is now ready to be run by hitting the Run button. Figure 45 shows where this is located on the overall Experimental frame Input panel.

Figure 45. Run Button Location

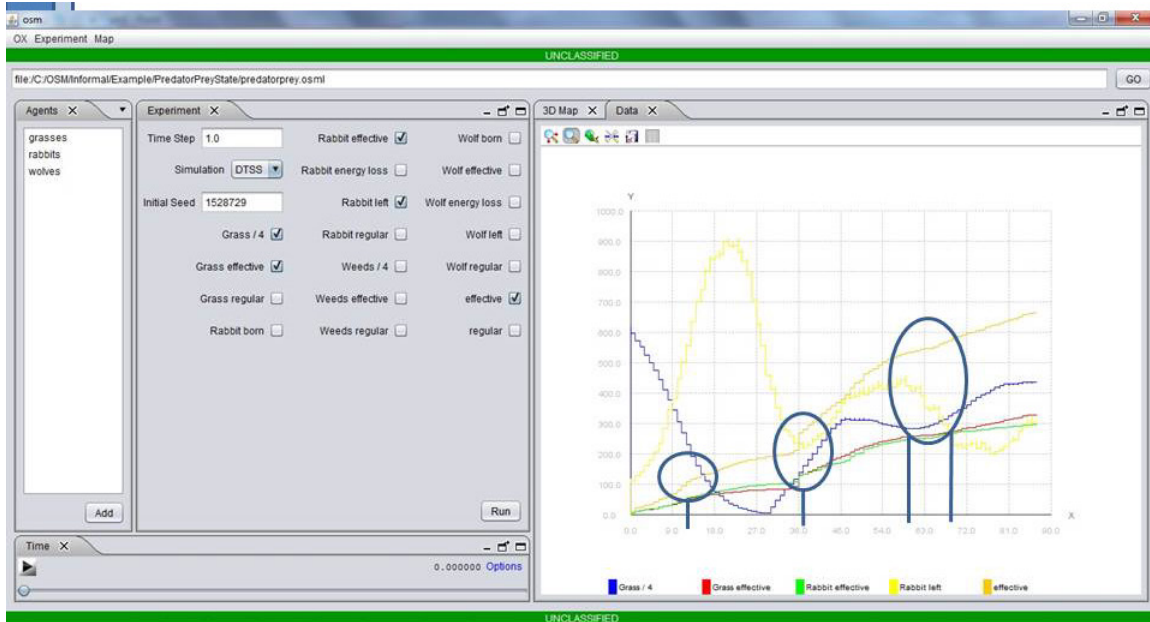


E. EMERGENT BEHAVIOR IDENTIFIED AND QUANTIFIED

Once the run or runs for the experiment are completed, the user will analyze the results to identify and quantify expected and unexpected emergent behavior. This is done through the graph provided with the simulation. Figure 46 shows the graph from the simulation (as displayed through the OSM framework) with all of the selected metrics displayed. This graph shows that changes in the interaction metrics slope along with changes in the agent metrics show emergent behavior. Using Predator Prey as an example, we see in the first highlighted area that the grass is reducing at a fast pace as the

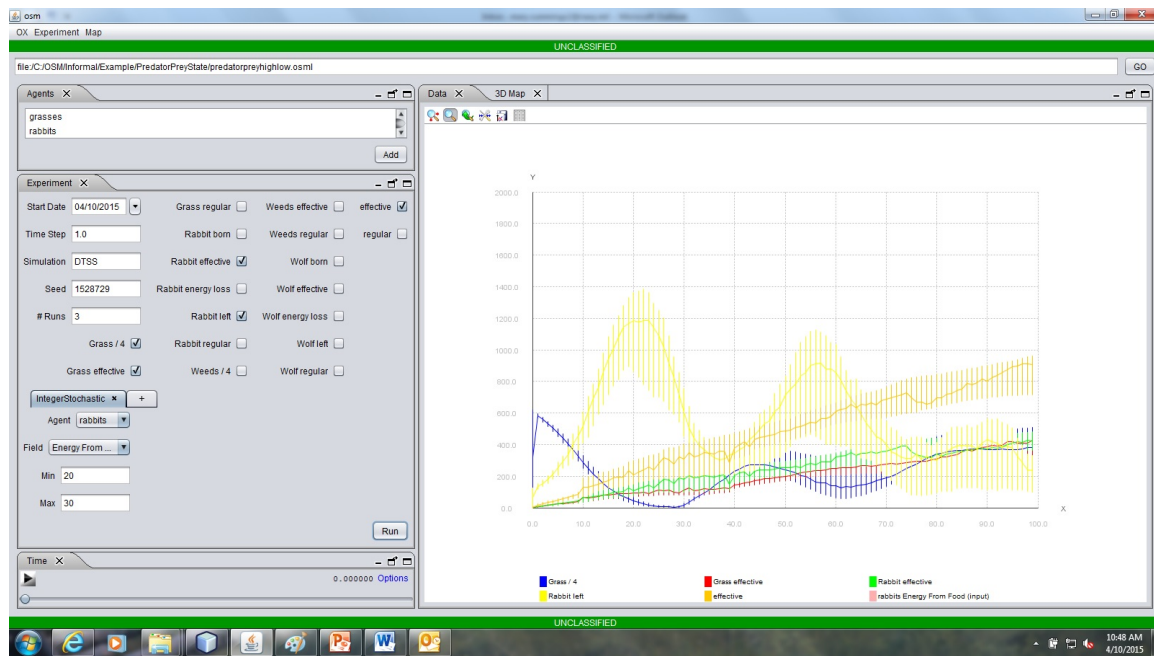
rabbits are increasing. Right after the lines cross, the slope of the overall effective interaction metrics changes.

Figure 46. Results of One Run of a Simulation



After the first set of runs, the user may realize that other agent and interaction metrics are needed in order to identify and quantify the emergent behavior for this set of models. In the Predator Prey example scenario, the user tried many combinations of metrics. It was found that the wolves did not make a difference to the metrics as a whole, but the grass and rabbits did. It was also found that the regular interaction metrics did not make a difference. From that, the user decided that those metrics could be removed from the graph to allow better viewing of the metrics that did make a difference. If the user chose to do multiple runs as part of a particular scenario, then the graph will show the median, maximum and minimum values at each timestep. This allows the user to see if there are any large swings in data at a particular time in the simulation. This could be a reason for the user to do extra analysis on why this data had such a large difference between runs. This large swing in data can also cause a change in the emergent behavior at that point in the simulation. See Figure 47 for an example of a multi run graph using the Stochastic Experimental frame.

Figure 47. Results of a Stochastic 3 Run Graph



V. PROOF OF CONTRIBUTIONS

To demonstrate that this research can truly provide a method for identifying and quantifying emergent behavior and provide swappable Experimental frames and Simulators, two simulations were developed to show the power of this work. Three Experimental frames and three Simulators were developed. We used each of these in both simulations to show that the same Experimental frame and Simulator can be used in both simulations, and to show that multiple Experimental frames and Simulators can be used in a simulation without changing the other parts of the simulation (namely, the models, also known as the agents). These two simulations are the Predator Prey simulation and the BMDS simulation.

This is done through layers of standard interfaces. We use Java interface classes and abstract classes to do this. These layers enable interoperability and swappability. These allowed us to provide a method for any simulation where emergent behavior in an SoS simulation can be identified through the collection of SoS metric data. We demonstrated this capability with the Predator Prey simulation. Both of these simulations were built using the OSM framework, as described in Chapter II.

A. PREDATOR PREY SIMULATION

In order to demonstrate various achievements in this research, five versions of the Predator Prey simulation are produced. Each of these (except for the baseline version listed first) was run (as part of this Proof of Concept) with all three Experimental frames. Also all (except for the baseline version listed first) were run with grass only as food for the rabbits and with grass and weeds both as food for the rabbits. In other words, 25 types of runs were made. Table 2 lists the versions, Simulator used, agents used, and Experimental frames run for each simulation. It must also be noted that the Grass and Weeds Agents did not change between any of the versions listed below.

Table 2. Types of Simulations Developed and Run

Version	Simulator	Move type	Rabbit Agent Used (Driven by Events)	Rabbit Agent Used (Driven by Time)	Wolf Agent Used (Driven by Events)	Wolf Agent Used (Driven by Time)	Grass Agent Used	Weeds Agent Used	Exp Frames Used
1) Netlogo	Discrete	random		X		X	X	X	Multi run
2) Discrete Event	Discrete	Closest	X		X		X	X	All 3
3) Discrete Time	Discrete	Closest		X		X	X	X	All 3
4) DEVS	DEVS	Closest	X		X		X	X	All 3
5) DTSS	DTSS	Closest		X		X	X	X	All 3

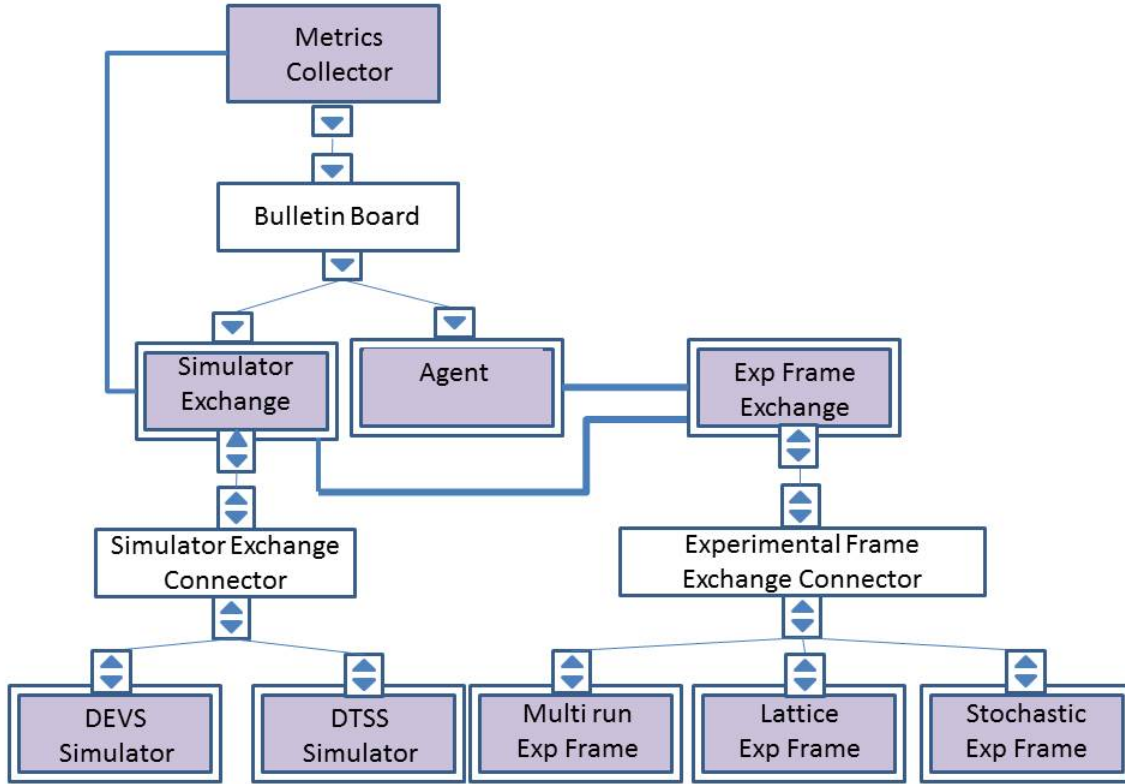
The versions are described as:

1. Matching the NetLogo wolves sheep simulation, using the discrete Simulator, the rabbits and wolves move randomly and are all driven by time, uses grass agent as food for the rabbit, and run with the Multi Run Simulator
2. Using the discrete Simulator, the rabbits and wolves move toward closest food and are all driven by discrete events, uses one or both grass and weeds as food for the rabbit and run with each of the three Experimental frames
3. Using the discrete Simulator, the rabbits and wolves move toward closest food and are all driven by discrete time, uses one or both grass and weeds as food for the rabbit and run with each of the three Experimental frames
4. Using the DEVS Simulator, the rabbits and wolves move toward closest food and are all driven by discrete events, uses one or both grass and weeds as food for the rabbit and run with each of the three Experimental frames.
5. Using the DTSS Simulator, the rabbits and wolves move toward closest food and are all driven by discrete time, uses one or both grass and weeds as food for the rabbit and run with each of the three Experimental frames.

The discrete Simulator was created such that events could be passed in or time steps could be passed in but in order to use the other method (for example, from events to time) the model had to be changed.

Figure 48 shows the architectures of all of the elements of this work (the Metrics Collector, the Experimental frame Exchange, and the Simulator Exchange) together. For the Predator Prey simulation, the agent block represents the Rabbits, Wolves, Grass and Weeds.

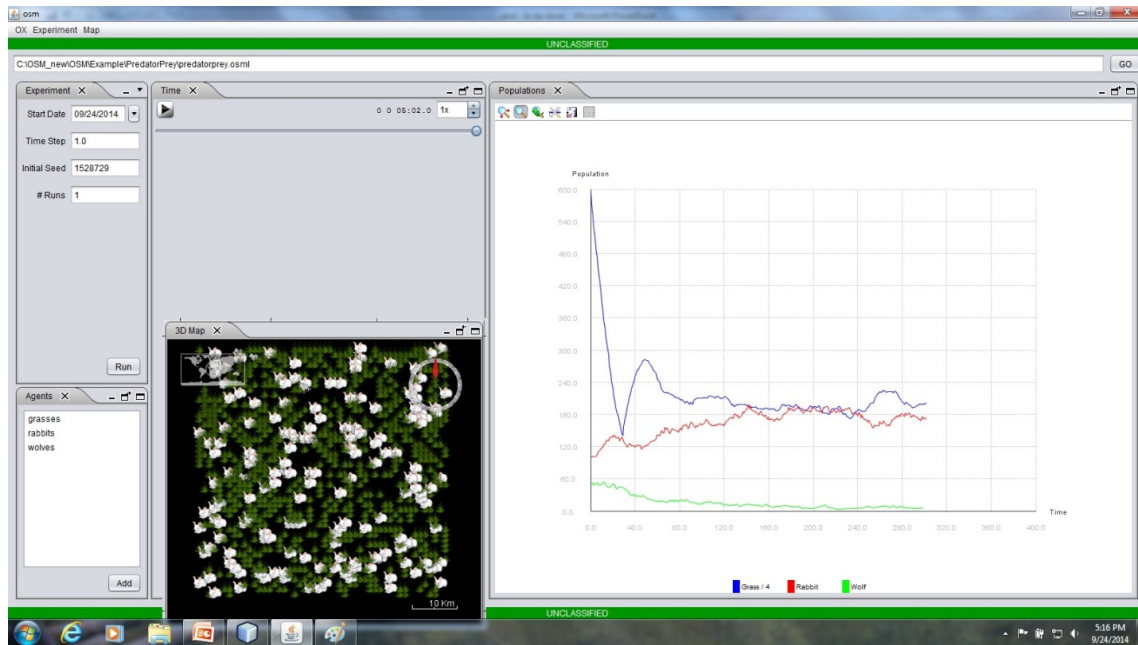
Figure 48. The Elements of This Research For this Proof of Concept



1. Description of NetLogo Matched Version

The purpose of this simulation is to provide a baseline implementation for the Predator Prey simulation. The Predator Prey simulation was built from the explanation of the Wolf Sheep Predation example (detailed in Wilensky (1997)) in the model library of NetLogo (Wilensky, n.d.). Our variation used rabbits instead of sheep. In this simulation, there is grass, rabbits, and wolves placed randomly in a 50 x 50 cell grid. Figure 49 shows the simulation shown in the OSM framework.

Figure 49. Start of Predator Prey simulation



The simulation includes three sets of agents—wolves, rabbits, and grass. The wolves eat rabbits and rabbits eat grass. All are placed randomly on a 50 x 50 cell canvas. The number of initial agents plotted on the canvas is input. The cells that do not have grass are blank (or black) patches that are capable of growing grass. All animals are initially given a number of units of energy (input). The animals move randomly around the canvas. Because they move to the next cell each timestep, this is really a Discrete Time simulation. For each movement, they lose one unit of energy. If it expends all of its energy before it reaches food, the animal dies. As each animal moves, it may reproduce based on a reproduction probability (input). If an animal lands on the same cell as its food, it eats it. With that, it gets some number of energy units (input). The eaten element is destroyed. The initial number of grass is chosen randomly. The other blocks on the canvas (black/blank) are provided a random time unit for regrowth. Once eaten, the grass is regrown using a regrowth time (input). There is some probability of reproduction (input) for each animal for each amount of distance it makes each second as it moves toward the food. Figures 50, 51, and 52 show the Finite State Automata that was the basis for this version. This FSA shows the states that the agent has and the transitions that force

the changes in state. As stated in the Implementation section for agents in Chapter III, this FSA is then used as the basis for implementing the agent.

Figure 50. Rabbit FSA

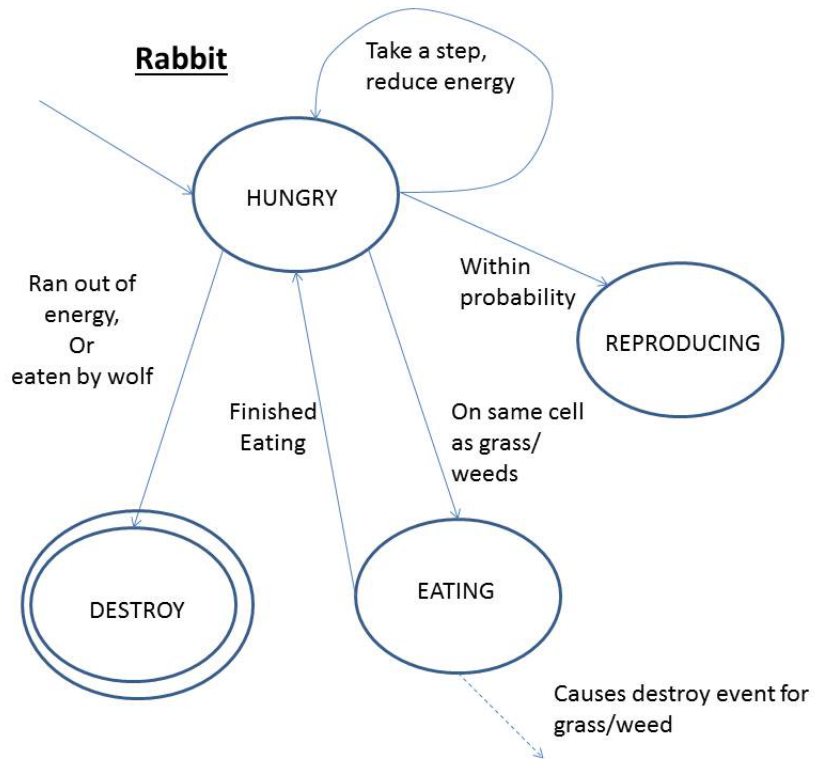


Figure 51. Wolf FSA

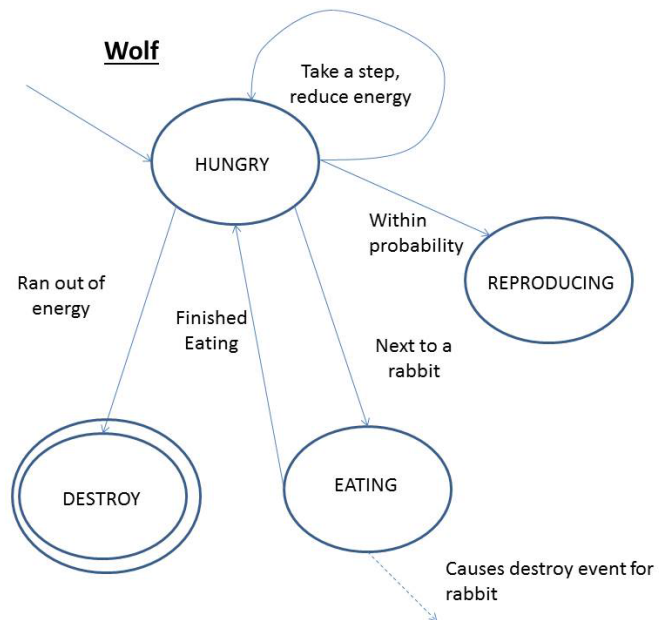


Figure 52. Grass FSA

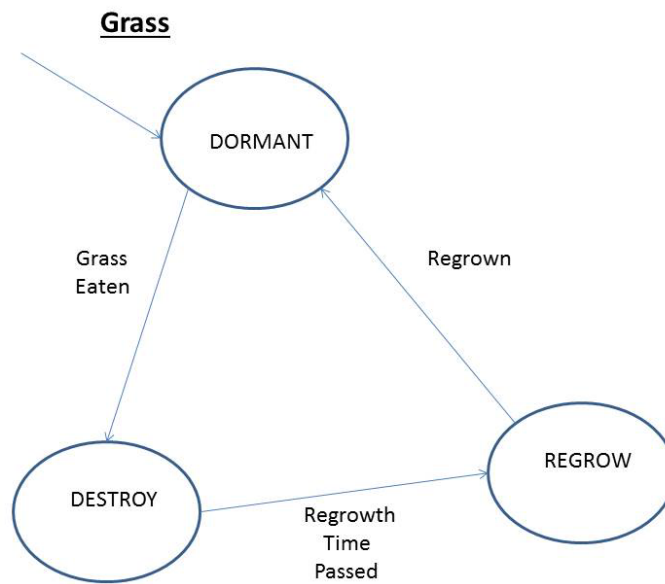
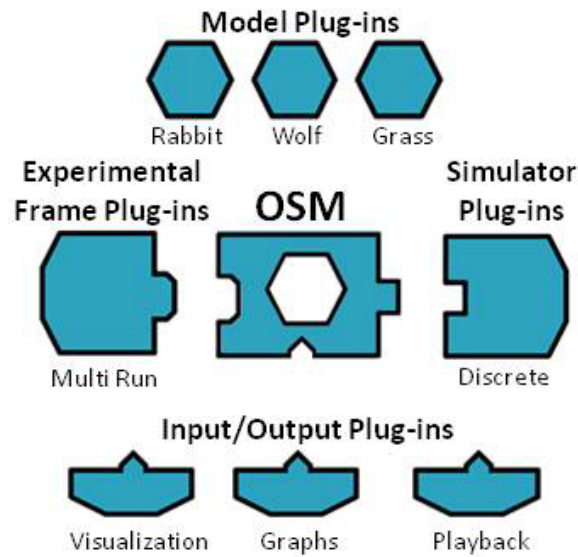


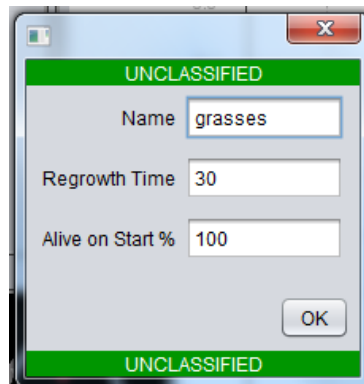
Figure 53 shows the software architecture for this simulation. It shows the three models (agents), the Multi Run Experimental frame, and the Discrete Simulator. It also shows that there were three output plug ins—Visualization (the 50 by 50 grid), graphs (the line graph) and the playback that allows the moving forward and backward in time of the simulation.

Figure 53. Input Parameters for the Grass



The Experimental frame used was the multi run Experimental frame. Figures 54, 55, and 56 show the input parameters for each of the three agents. This version was built using the Discrete Simulator.

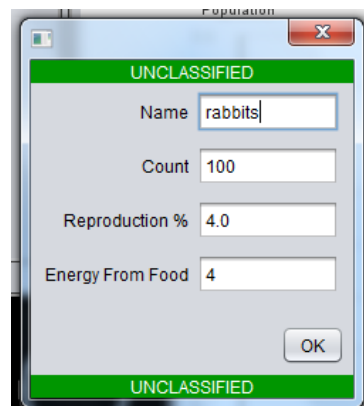
Figure 54. Input Parameters for the Grass Agent



A dialog box titled "UNCLASSIFIED" with a green header and footer. It contains three input fields: "Name" with the value "grasses", "Regrowth Time" with the value "30", and "Alive on Start %" with the value "100". An "OK" button is located at the bottom right.

Parameter	Value
Name	grasses
Regrowth Time	30
Alive on Start %	100

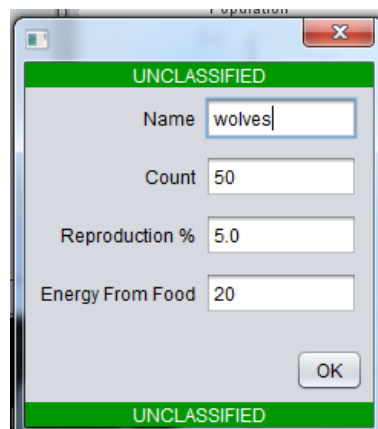
Figure 55. Input Parameters for the Rabbit



A dialog box titled "UNCLASSIFIED" with a green header and footer. It contains four input fields: "Name" with the value "rabbits", "Count" with the value "100", "Reproduction %" with the value "4.0", and "Energy From Food" with the value "4". An "OK" button is located at the bottom right.

Parameter	Value
Name	rabbits
Count	100
Reproduction %	4.0
Energy From Food	4

Figure 56. Input Parameters for the Wolf Agent

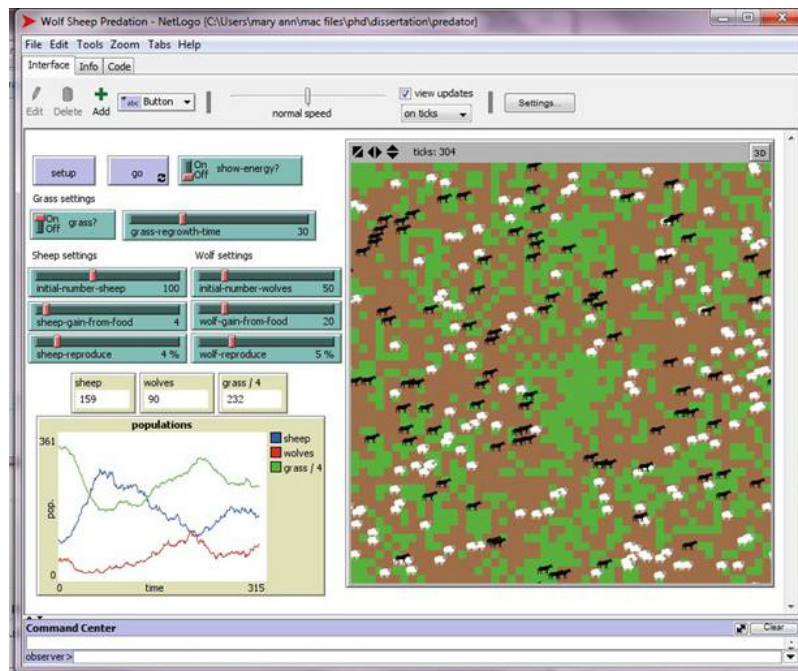


A dialog box titled "UNCLASSIFIED" with a green header and footer. It contains four input fields: "Name" with the value "wolves", "Count" with the value "50", "Reproduction %" with the value "5.0", and "Energy From Food" with the value "20". An "OK" button is located at the bottom right.

Parameter	Value
Name	wolves
Count	50
Reproduction %	5.0
Energy From Food	20

Figure 57 shows a picture of a run of the NetLogo Predator Prey simulation. The panel has the inputs to the left, the resulting graph of the run at the bottom left and the visualization (that changes as the simulation is run) to the right of the panel. In this version, sheep are used in place of rabbits, but the rabbits and sheep are identical in their behavior.

Figure 57. NetLogo Predator Prey Simulation



This version has a 50 by 50 grid colored brown where there are no grass, sheep or wolves on it. The movement of the sheep and wolves are not entirely random. Each agent moves in a direction. In choosing its next cell movement, it is limited to a certain angle in picking the next cell as described in Section 2.9.1.2.

This same (better to call it similar) version run using the OSM framework with our swappable Experimental frame and swappable Simulator, we have run varying the inputs to see the different results. We ran it multiple times because we could not match the NetLogo results exactly (as discussed in section 2.9.1.2). Figures 58, 59, and 60 show three runs and results. These figures show the inputs used to generate this run. They also

show the NetLogo graph overlayed on the screen shot to show the difference in the two runs.

Figure 58. Run 1 using OSM Framework with Random Movements to Match NetLogo Run

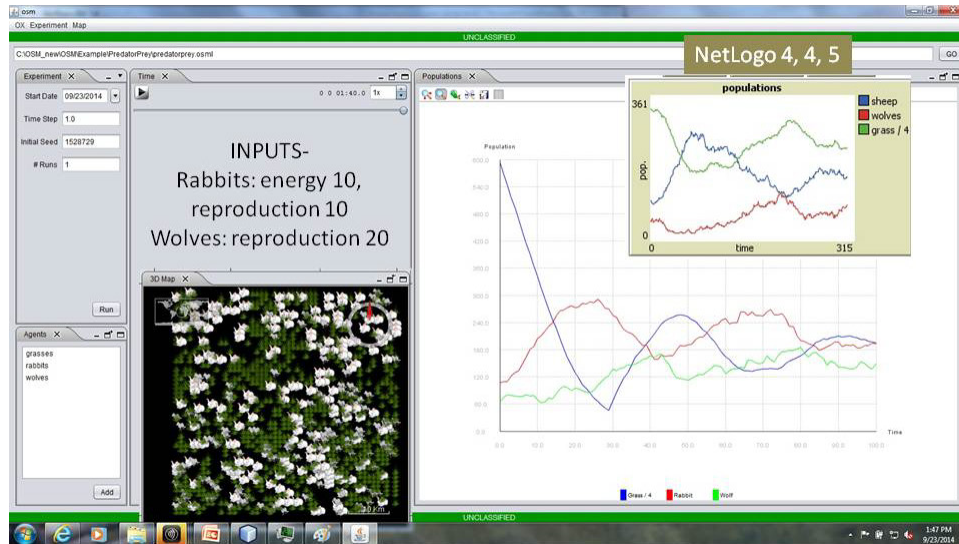


Figure 59. Run 2 using OSM Framework with Random Movements to Match NetLogo Run

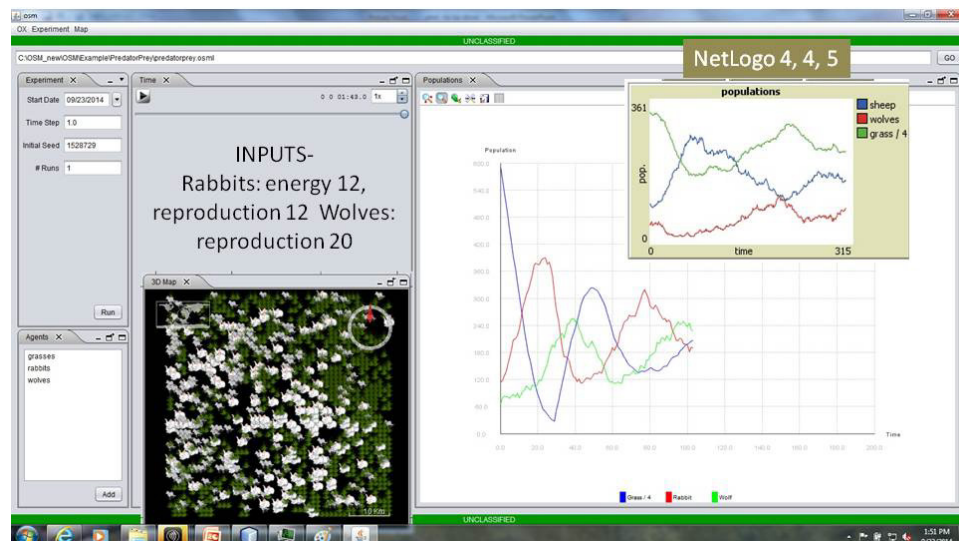
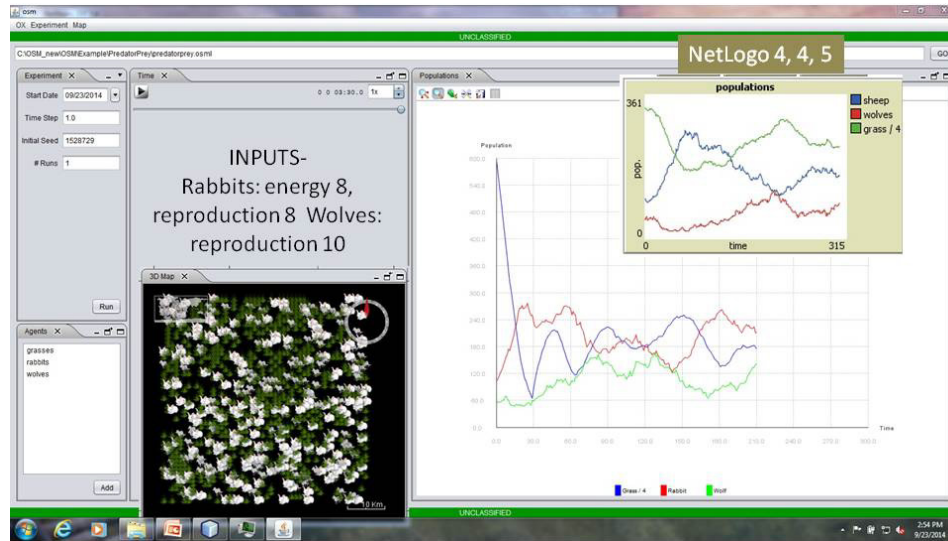


Figure 60. Run 3 using OSM Framework with Random Movements to Match NetLogo Run



2. Comparison of NetLogo and OSM

The reasons for the differences in the plots can be many. One significant difference is the number of events generated per time tick. For NetLogo, there is one event per tick. For the Predator Prey simulation in osm, there are on average 21 events per time tick for two rabbits and two wolves. For the first run shown with 100 rabbits and 50 wolves, there are on average 211 events per time tick. For NetLogo, the size of the simulation space is set to 50 by 50. Bajrachaya and Duboz (2013) tells us that agent-based platforms (like NetLogo and OSM) may be biased by design and implementation choices which leads to the inability to compare results. The authors used cross correlation testing to see if there was a similarity in shape between the curves in output plots between the three different simulation platforms: NetLogo, Cormas and Repast. The authors created the same Agent Based Model (SIR) on these three platforms. They used a cross-correlation coefficient function and place a threshold on the two values to determine if the two time series are similar. They found that the Probability of Similarity (POS) among the platforms decreased as the threshold increased. The authors then determined from this data that the inability “to control the scheduling of events is one of the major factors for the variation in results from these three platforms” (Bajrachaya & Duboz,

2013). This tells us that our inability to compare the NetLogo and OSM versions of the Predator Prey simulation is expected.

The NetLogo version has a set size of the canvas. It appears to be 50 by 50. The initial number of grass is chosen randomly. The other blocks on the canvas are colored brown. After the grass is eaten, the grass is regrown after 30 time steps (as in the OSM simulation). The brown patches that are shown initially also have this same regrowth attribute set. With only two sheep and two wolves, this provides a greater probability that the brown patches will have time to regrow into grass because there are very few sheep to eat the grass so it is not reducing faster than the brown patches are regrowing into grass in the beginning of the simulation. This means that even though there may be 1,248 initial patches of grass (as shown in Figure 34), there could be as many as 2,500 early in the simulation before the sheep start to reproduce.

The NetLogo version allows the sheep and wolves to move randomly around the canvas. If one of the sheep lands on a patch of grass or one of the wolves lands on a sheep, then it eats it. The OSM version has each rabbit or wolf move toward its closest food source. It eats the food source when it is within some tolerance of distance. The rabbit and wolves only move to attain food.

The NetLogo version randomly picks the amount of grass to start. The NetLogo version does not allow the setting of the grass so we used 1,312 as the grass elements to match NetLogo's random pick of grass for a particular run.

3. Simulators Used in the Closest Food Version of the Predator Prey Simulation

a. Discrete Event Version Using the Discrete Simulator

These agents in this version were written to be driven by Discrete Events. In order to do that, instead of moving randomly, each animal agent moves toward the closest food. This version was based on the Finite State Automaton (FSA) shown in Figure 61. Only the Wolf and Rabbit FSA is shown here because the Grass FSA is the same for the random version and the closest food version. Figure 62 shows the visualization and the graph of this version with the Rabbits agents input parameters changed to energy set to

20, and reproduction set to 20 percent. The changes for the Wolves agents input parameters are: number of wolves set to 20, energy set to 8, and reproduction set to 8.

Figure 61. Wolf and Rabbit FSA

Wolf/Rabbit

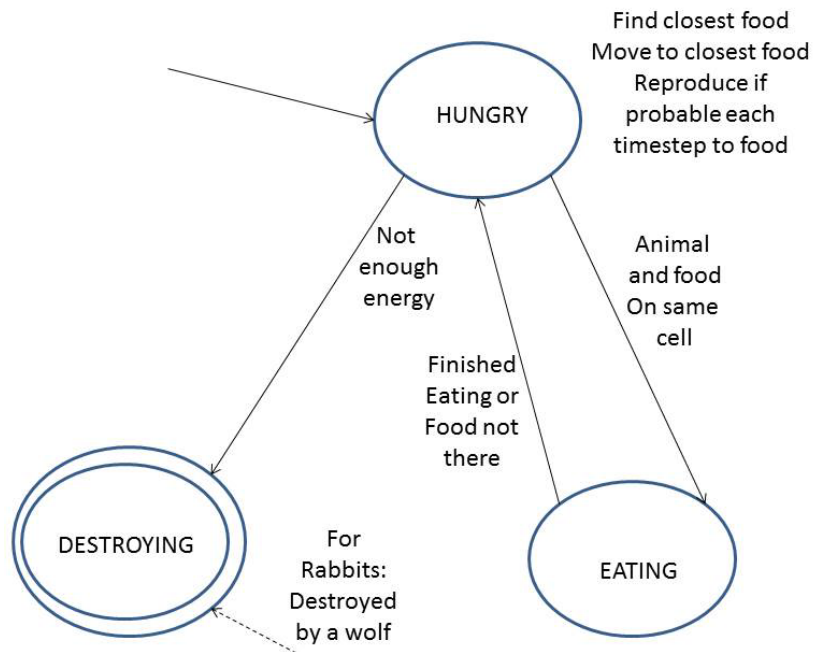
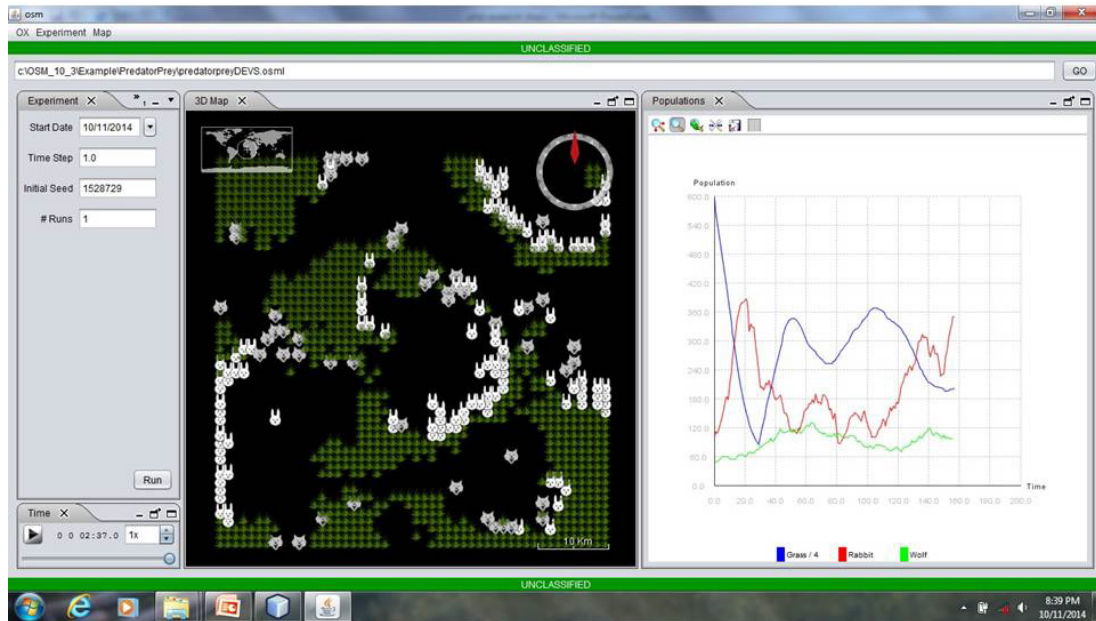


Figure 62. Simulation Based on FSA



b. Discrete Time Version Using the Discrete Simulator

This version was for each animal to be driven by Discrete Time and to move toward the closest food. It goes through each event for each agent each time step. For this simulation, the time step is set to 1 second. It does not follow Zeigler's DTSS formalism because it does not have states, only events. This version was based on the FS-TA shown in Figures 63, 64, 65 and 66. This FS-TA shows the states that the agent has and the transitions that force the changes in state. As stated in the Implementation section for agents in Chapter III, this FS-TA is then used as the basis for implementing the agent.

Figure 67 shows the visualization and the graph of this version with the Rabbits agents input parameters changed to energy set to 20, and reproduction set to 10 percent. The changes for the Wolves agents input parameters are: number of wolves set to 25, energy set to 5, and reproduction set to 20. One change for the Grass agents input parameters was made: regrowth set to 15 time steps. We used different parameters for the Discrete Time version was because we wanted to find parameters that would cause the simulation to oscillate vice cause one of the set of agents go to zero.

Figure 63. Wolf and Rabbit Discrete Time FS-TA

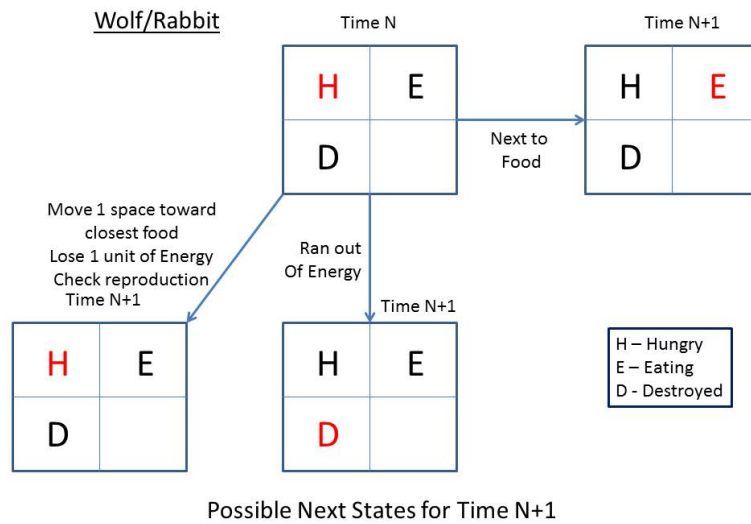


Figure 64. Wolf and Rabbit FS-TA

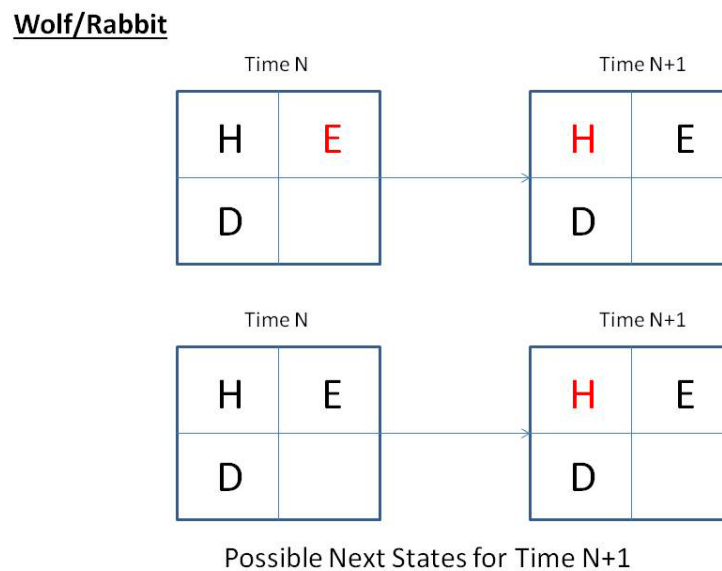


Figure 65. Grass FS-TA

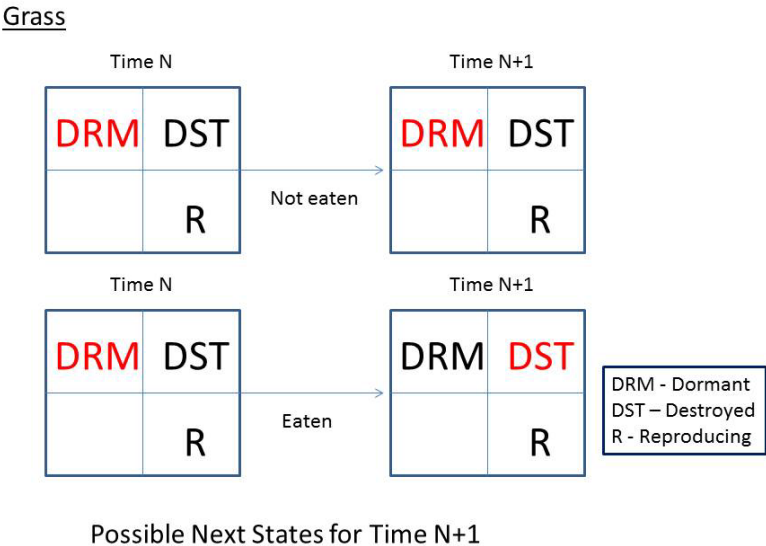


Figure 66. Grass FS-TA

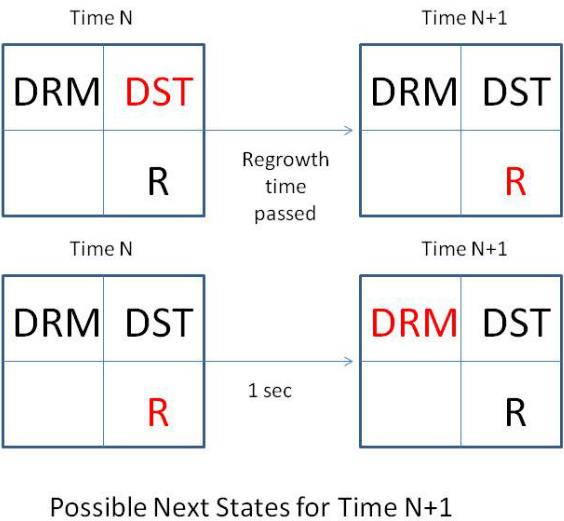
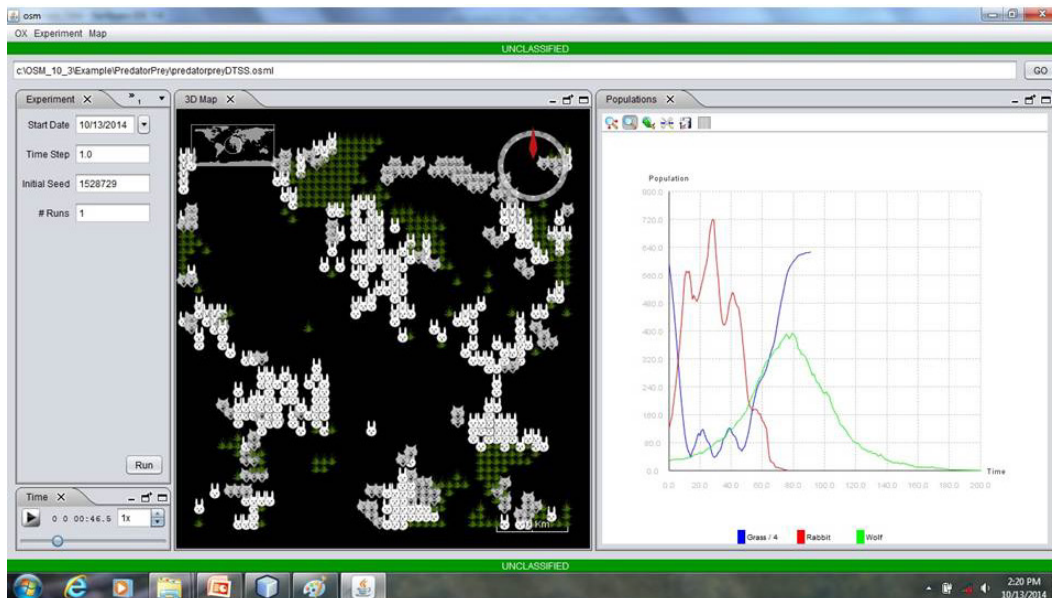


Figure 67. Simulation Based on FS-TAs



4. Addition of a Weed Agent

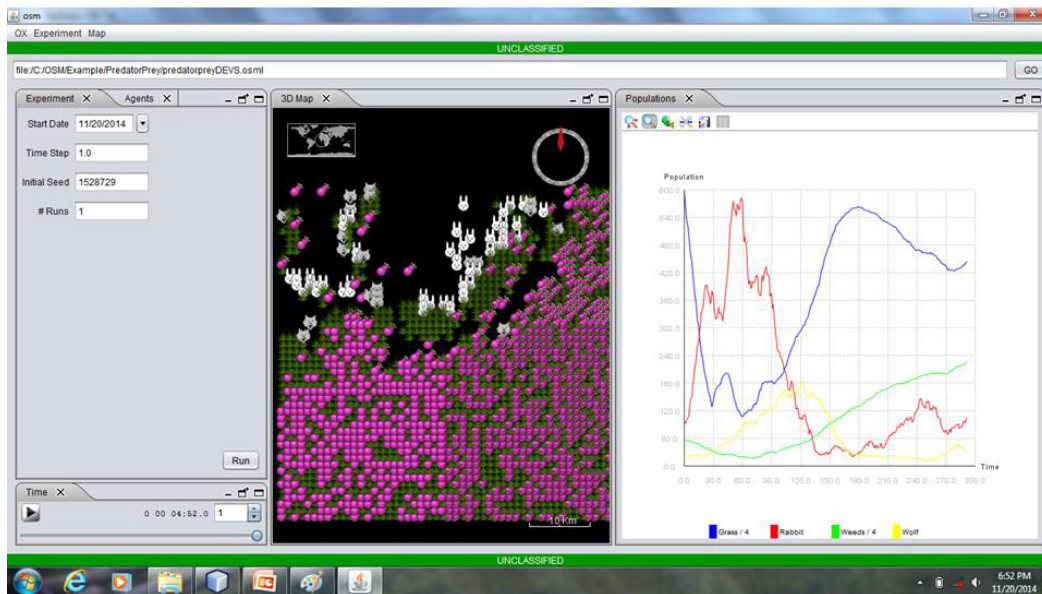
A weed agent was added to Discrete Event and Discrete Time versions of the Predator Prey simulation to show the changes that occur by adding a new agent without changing the rest of the agents, the Simulator, the Experimental frame, or any of the display plug ins. To understand how this is accomplished, see the Agent Implementation section in Chapter III. The rabbit will eat either grass or weed, whichever is nearest. If a rabbit eats a weed, the rabbit will die. Figure 68 shows the input parameter panel (as called by the Experimental frame Exchange) for the Weeds agent.

Figure 68. Weeds Input Parameters Panel

The screenshot shows a small dialog box titled 'UNCLASSIFIED' with a close button (X) in the top right corner. The dialog contains three input fields: 'Name' with the text 'weeds' entered, 'Regrowth Time' with the value '300', and 'Alive on Start %' with the value '10'. An 'OK' button is located at the bottom right of the dialog. The dialog box has a green header and footer bar, both labeled 'UNCLASSIFIED'.

Figure 69 shows the results of a run of this simulation using the same parameters for the other agents (Rabbits, Wolves and Grass) as previously shown with the Weeds input parameters set as above. Figure 70 shows the differences in the graph with a run using the same parameters for the other agents and changing the parameters for the Weed agent as shown.

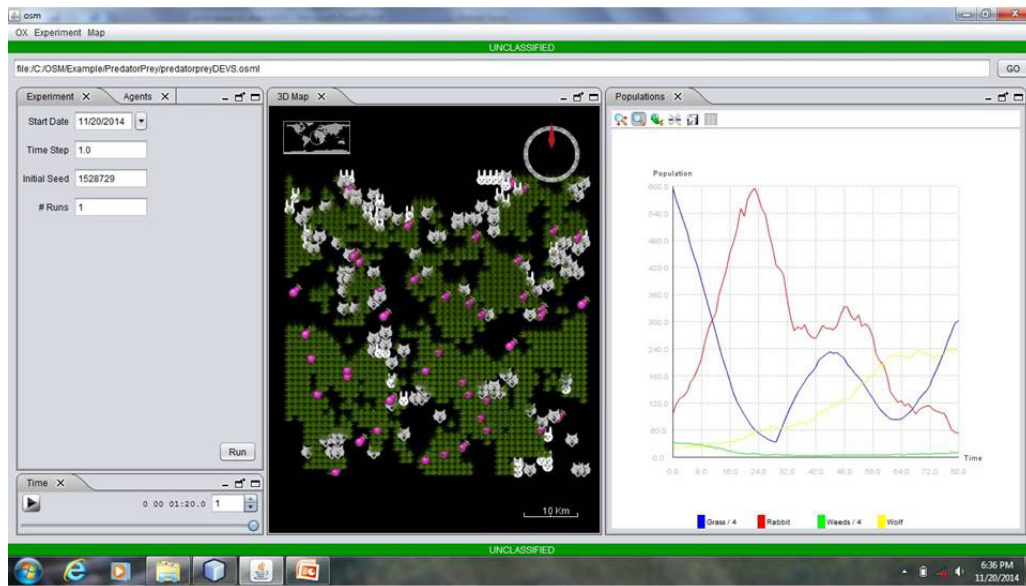
Figure 69. Results of Simulation with Addition of Weeds Agent



Used same settings for Rabbits and Wolves: Rabbits: energy – 20, repro – 20, Wolves: num: 20, energy – 8, repro – 8, and Weeds: Regrowth time – 300, Alive % - 10

58

Figure 70. Results of Simulation with Changes to Weeds Agent Parameters



Used same settings for Rabbits and Wolves: Rabbits: energy – 20, reproduction – 20, Wolves: num: 20, energy – 8, reproduction – 8, and Weeds: regrowth time – 1000, Alive % - 5

5. Simulation Driven by DEVS Simulator

This version and the next version were written to follow Zeigler's M&S theory, and to work for both the DEVS and DTSS simulators. The output is different because the user chose DEVS as the Simulator type. We chose the method of having the animals move toward the closest food source, instead of moving randomly, because it has a larger event set. Moving randomly and landing on a cell that is shared with food is time driven, vice event driven. This version was based on the Finite State Automaton (FSA) shown in Figures 51 and 60. This version uses the classes and methods shown in Chapter III. Each rabbit and wolf agent determines the closest food and schedules an event in the number of timesteps it takes to move to that food. Figure 71 shows the input parameters and the initial view of this version. The input parameters for the Rabbits are changed to set energy to 20, and reproduction to 20 percent. The changes for the Wolves agents input parameters are: number of wolves set to 20, energy set to 8, and reproduction set to 8. The grass input parameters did not change. Figure 72 shows the graph of the simulation, with the interaction metrics on the same graph as the SoS metrics. Figure 73 shows the visualization of the end of the simulation.

Figure 71. DEVS Simulation at Startup

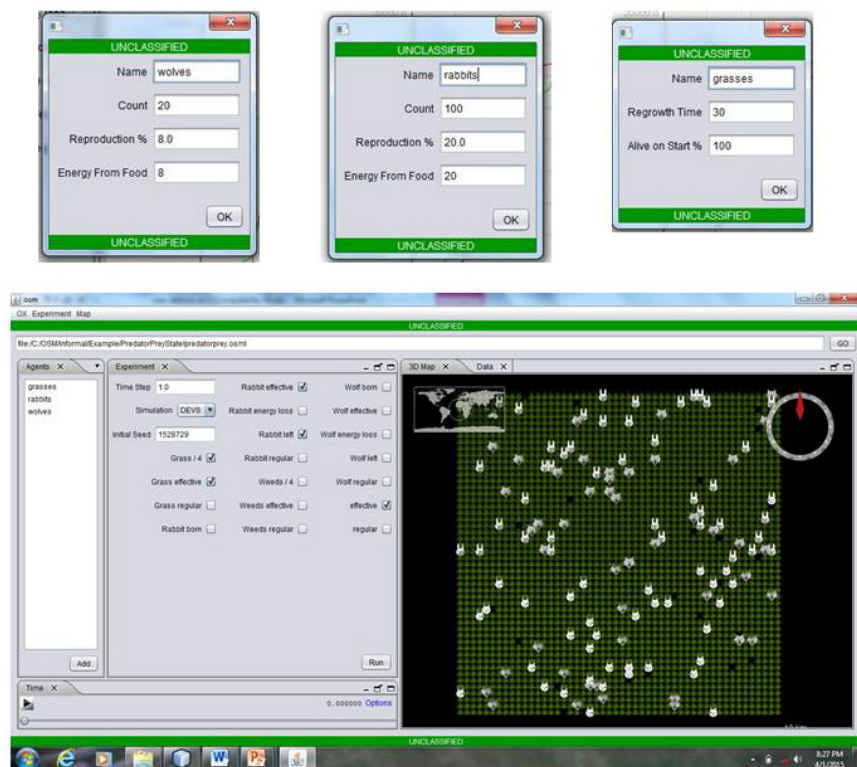


Figure 72. Graph of Results of DEVS Simulation

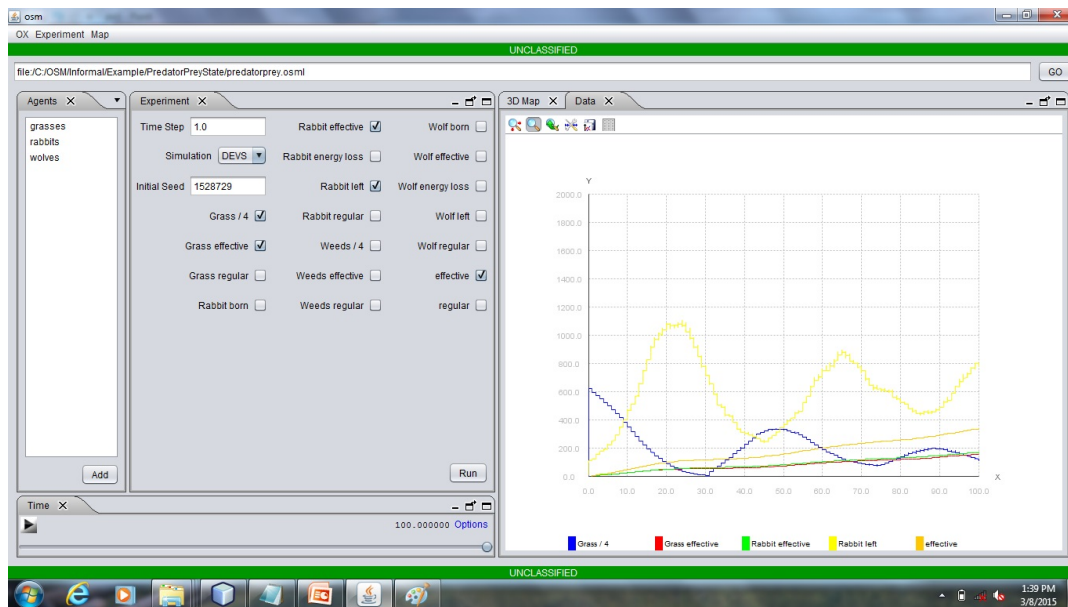
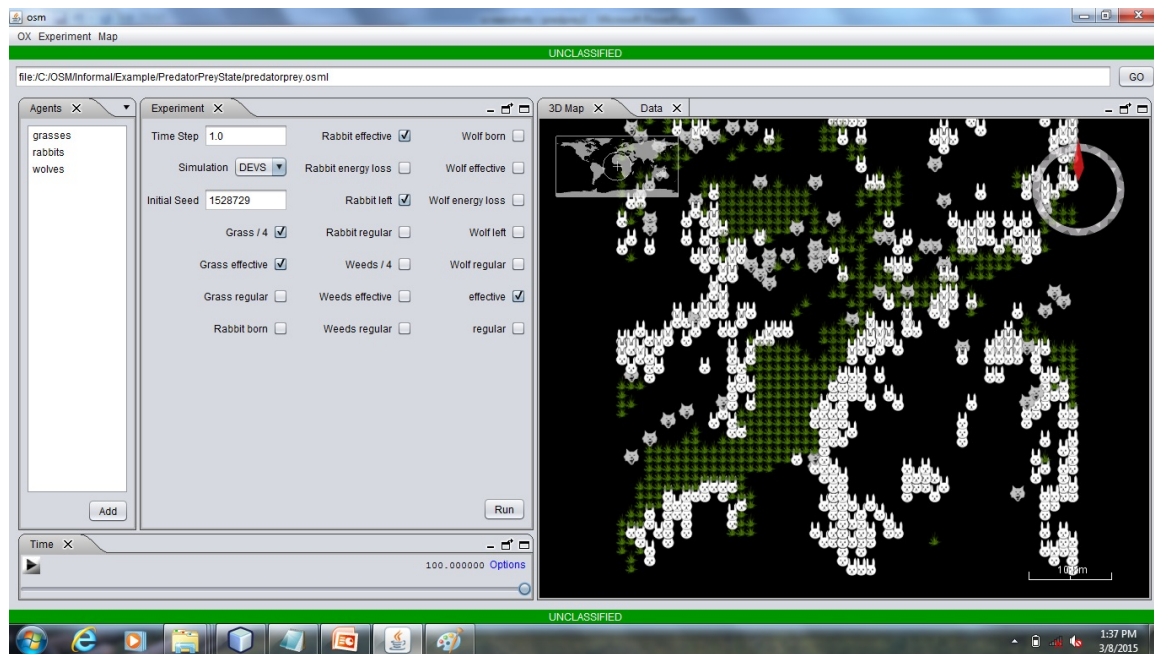


Figure 73. Visualization at end of DEVS Simulation



6. Simulation Driven by DTSS Simulator

This version and the previous version were written to follow Zeigler's M&S theory, and to work for both the DEVS and DTSS simulators. The output is different because the user chose DTSS as the Simulator type. This simulation has the animals move toward closest food. This version was based on the Finite State Automaton (FSA) shown in Figures 51 and 60, but the DTSS simulator drives this like the FS-TA in Figures 62–65. This version also uses the classes and methods shown in Chapter III. Because these agents did not change due to the Simulator change, each rabbit and wolf agent (as in the previous section) determines the closest food and schedules an event in the number of timesteps it takes to move to that food. But, in this simulation, the Simulator checks each state at every time step for a change in states (instead of waiting till the event occurs). Figure 74 shows the input parameters and the initial view of this version. The input parameters for the Rabbits are changed to set energy to 20, and reproduction to 20 percent. The changes for the Wolves agents input parameters are: number of wolves set to 20, energy set to 8, and reproduction set to 8. The grass input parameters did not change. Figure 75 shows the graph of the simulation, with the interaction metrics on the

same graph as the SoS metrics. Figure 76 shows the visualization of the end of the simulation.

Figure 74. DTSS Simulation at Startup

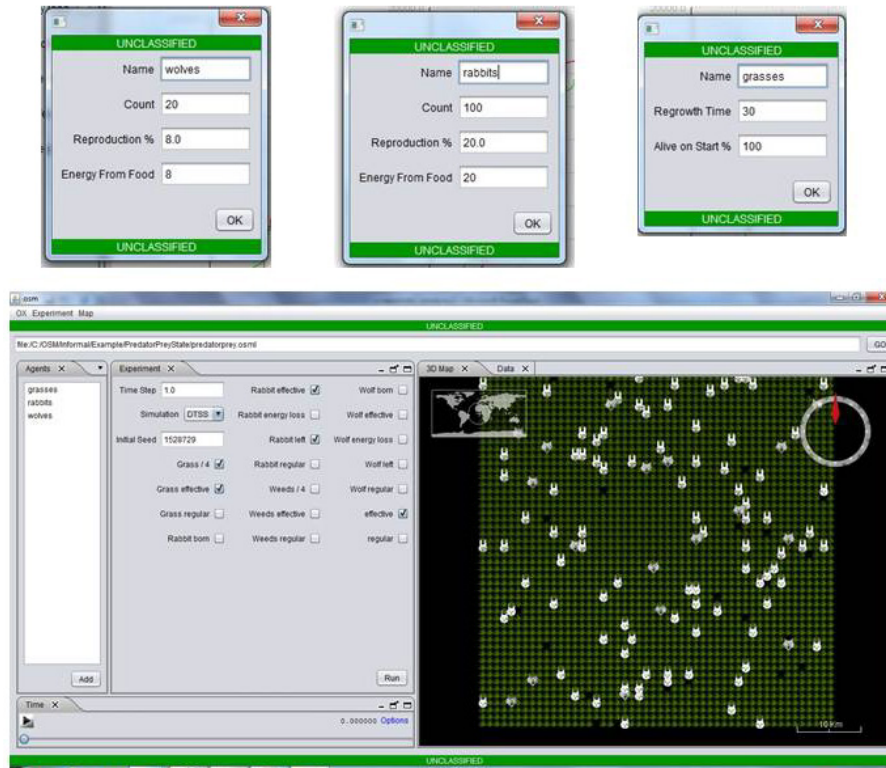


Figure 75. Graph of Results of the DTSS Simulation

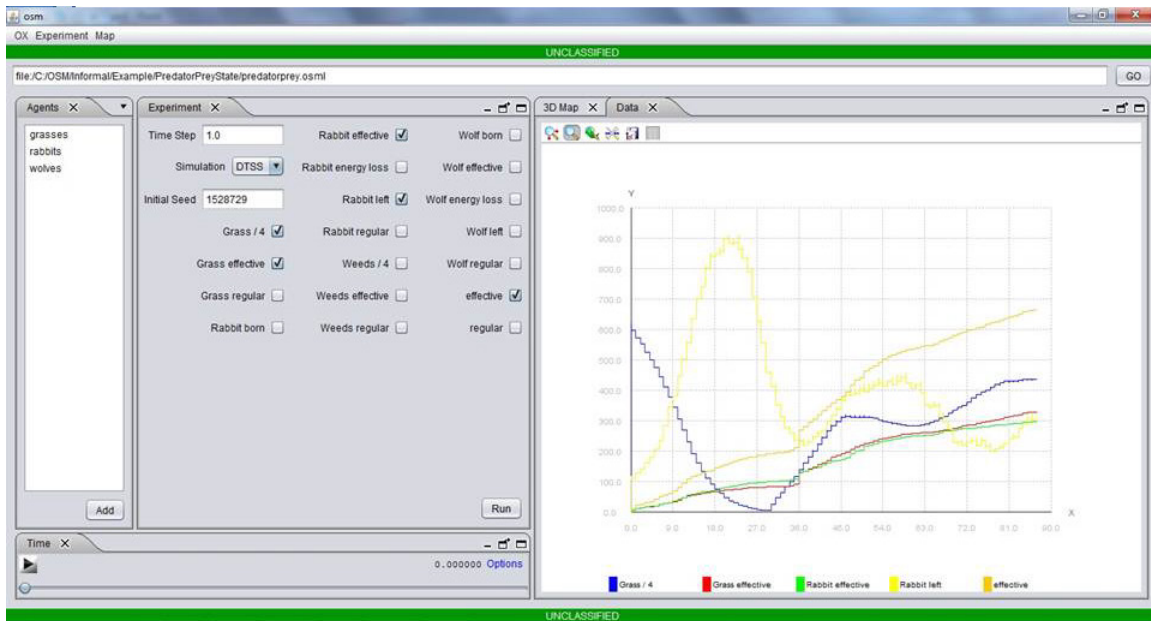
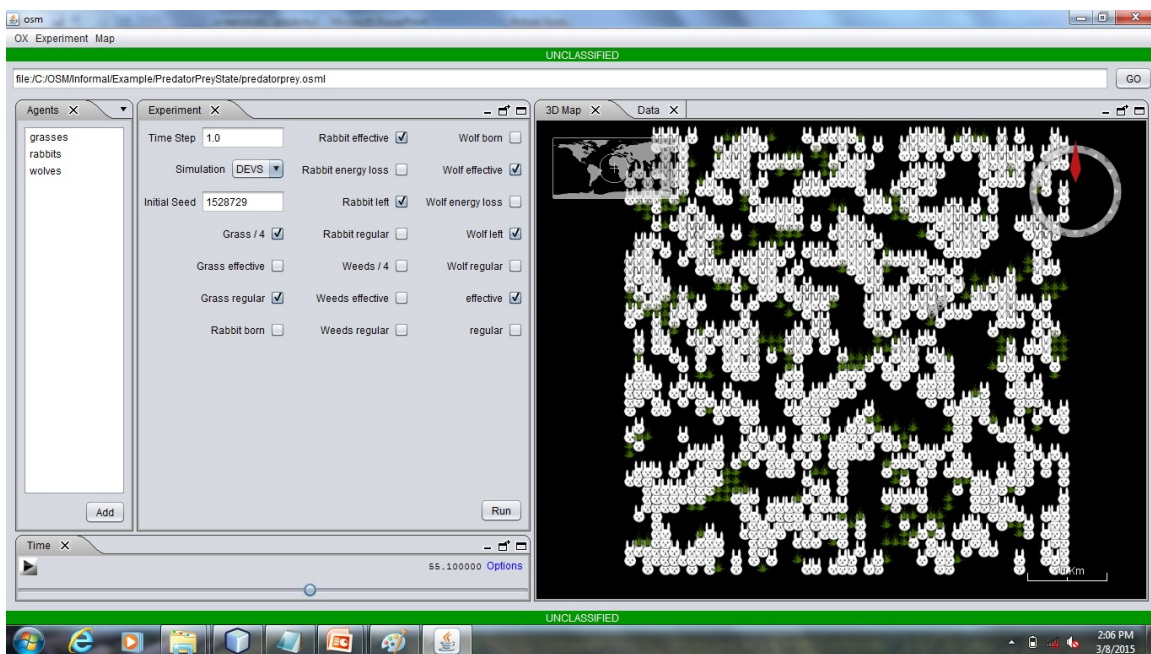


Figure 76. Visualization at end of DTSS Simulation



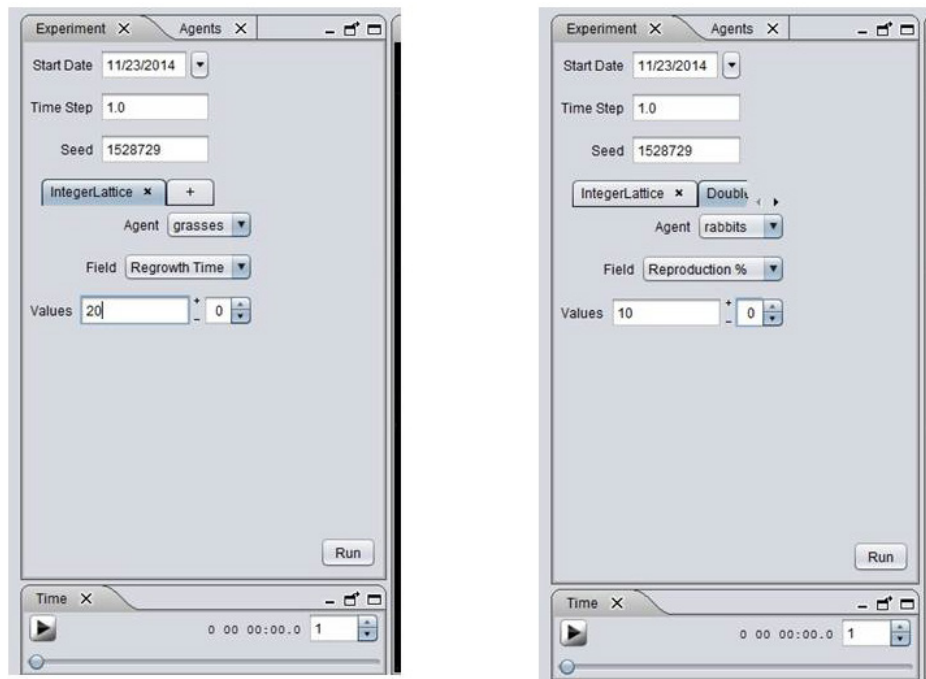
7. Simulation Driven by Discrete Simulator and Using the Lattice Experimental Frame

A Lattice Experimental frame is one in which one or more input parameters can be set as a collection of parameters. The Experimental frame will then determine the number of simulation runs by multiplying the number of each collection together. For example:

- Set 2 parameters, A and B, with three values each.
- $A(0) = 1, A(1) = 2, A(2) = 3$ and $B(0) = 10, B(1) = 11, B(2) = 12$
- The number of runs will be $3 \times 3 = 9$
- The values for the runs will be:
 - Run 1: $A=1, B=10$
 - Run 2: $A=1, B=11$
 - Run 3: $A=1, B=12$
 - Run 4: $A=2, B=10$
 - Etc

Figure 77 shows two input parameter panels using the lattice Experimental frame. In this scenario, the grass regrowth time is set to 20 and 30 time steps; the rabbit reproduction percentage is set to 10 and 20 percent.

Figure 77. Lattice Experimental frame Input Panels



Figures 78 and 79 show the graphs of the resulting data of the four runs. These graphs were built using Microsoft Excel vice using the graph plug in because we wanted to plot the data for the four runs together.

Figure 78. Graph of Each Agent Comparing Each Run

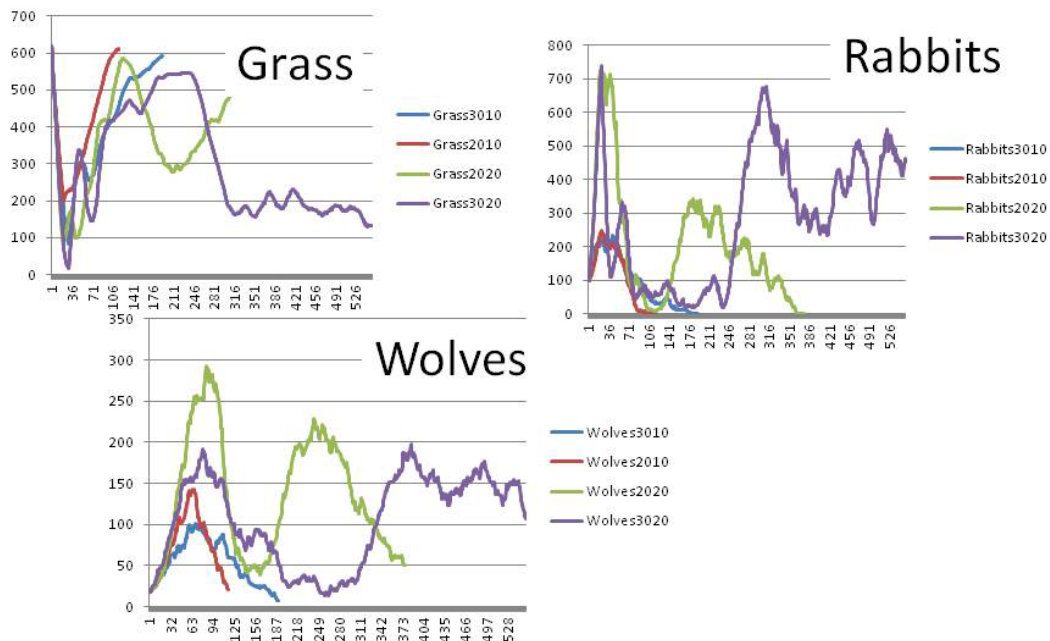
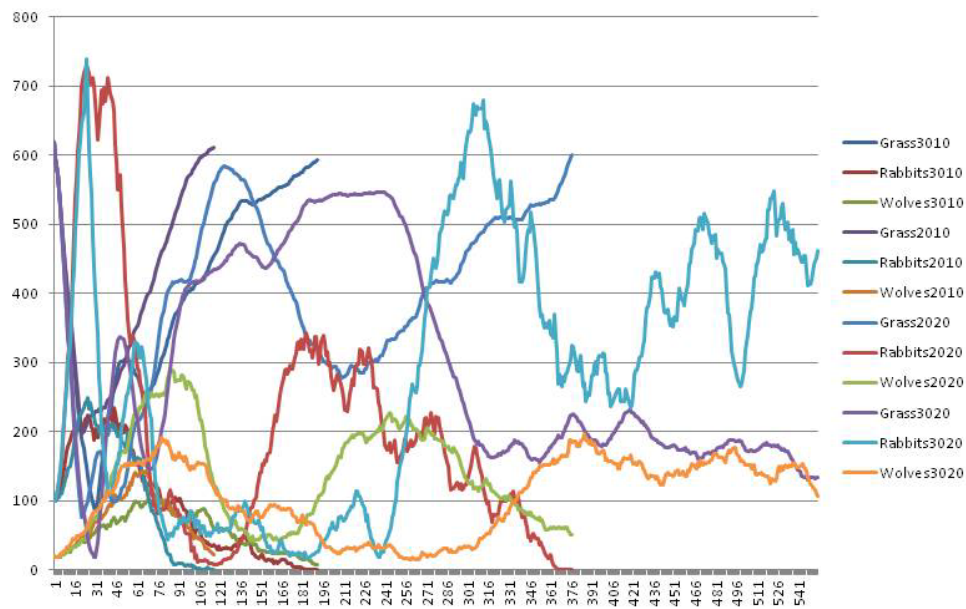


Figure 79. Graph of All Agents for All Runs



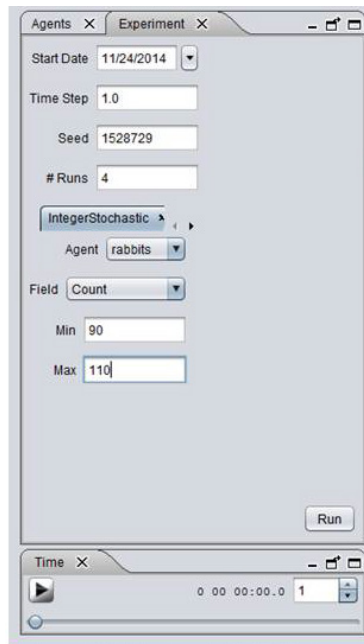
8. Simulation Driven by Discrete Simulator and Using the Stochastic Experimental Frame

A stochastic Experimental frame is one in which one or more input parameters can be set randomly between two limits. The user sets the number of runs in the Experimental frame. For example:

- Set 1 parameters, A, between two numbers
- Min value of 1 and max value of 10
- The user sets the number of runs to 3
- The Experimental frame uses a randomization function for each of the three runs to pick a value between 1 and 10

Figure 80 shows the input parameter panel using the stochastic Experimental frame. In this scenario, 4 runs are to be made using a rabbit initial count between 90 and 110.

Figure 80. Stochastic Experimental frame Input Panel



Figures 81 and 82 show the graphs of the resulting data of the four runs. These graphs were built using Microsoft Excel vice using the graph plug in because we wanted to plot the data for the four runs together.

Figure 81. Graph of Each Run

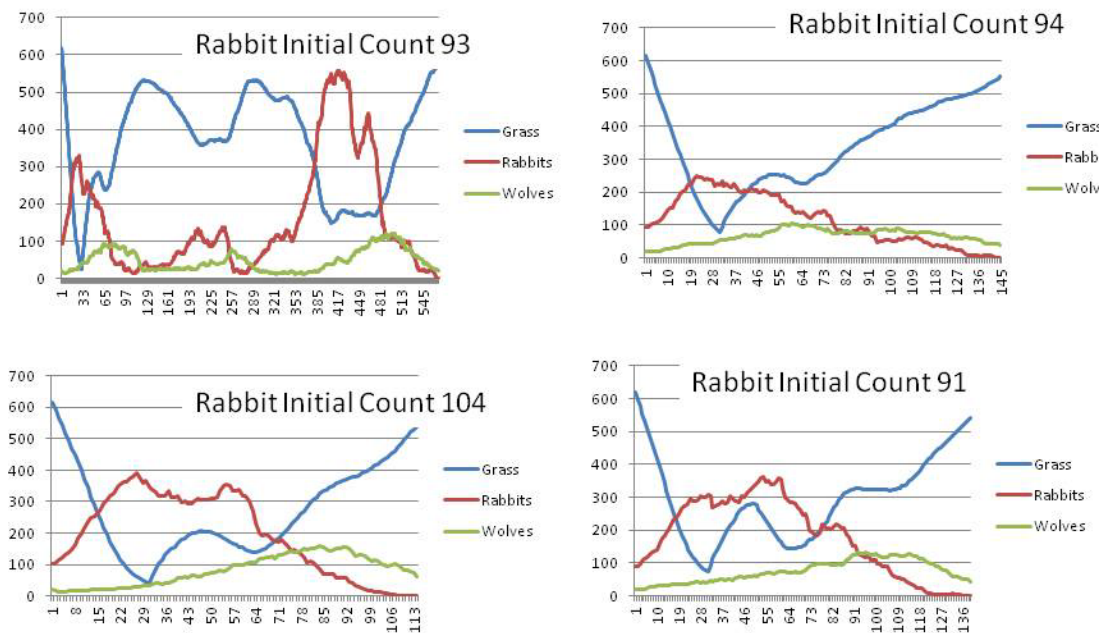
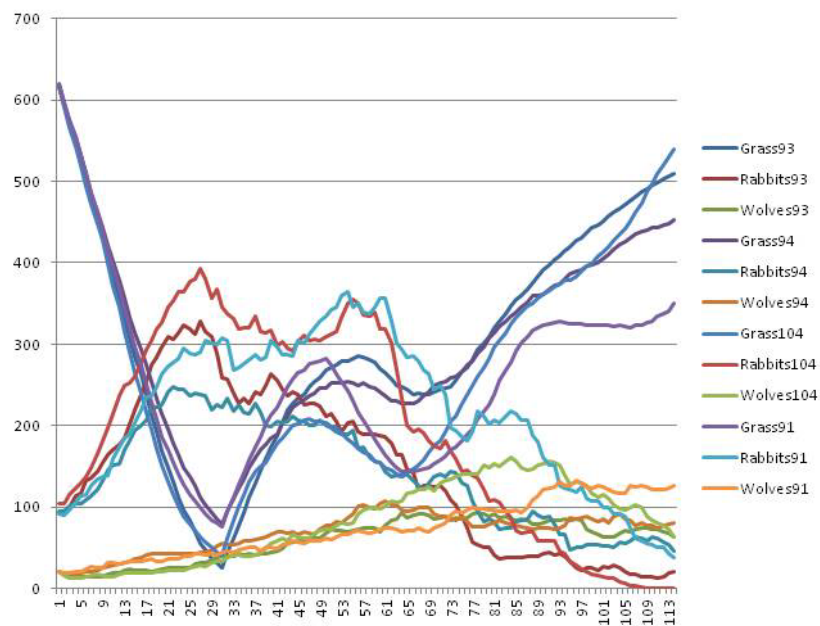


Figure 82. Graph Showing Results of All Four Stochastic Runs



B. EMERGENT BEHAVIOR IN PREDATOR PREY SIMULATION

We found through the execution of the Predator Prey simulation that we could identify and quantify emergent behavior. Figures 83 and 84 show the graph of the metrics (both interaction and agent metrics) through time (for both DTSS (Figure 83) and DEVS (Figure 84)). The circles were used to highlight the change in the interaction metrics (in the slope of the line) at various points in the timeline. We found that the effective interactions were the interaction metrics that showed interesting changes. These changes coincided with changes in the agent metrics. For example, in the first highlighted area for the DTSS run, the rabbits were increasing quickly while the grass was decreasing quickly. At the same time, the effective interactions showed a change in the slope.

Figure 83. Graph of Metrics for DTSS Run

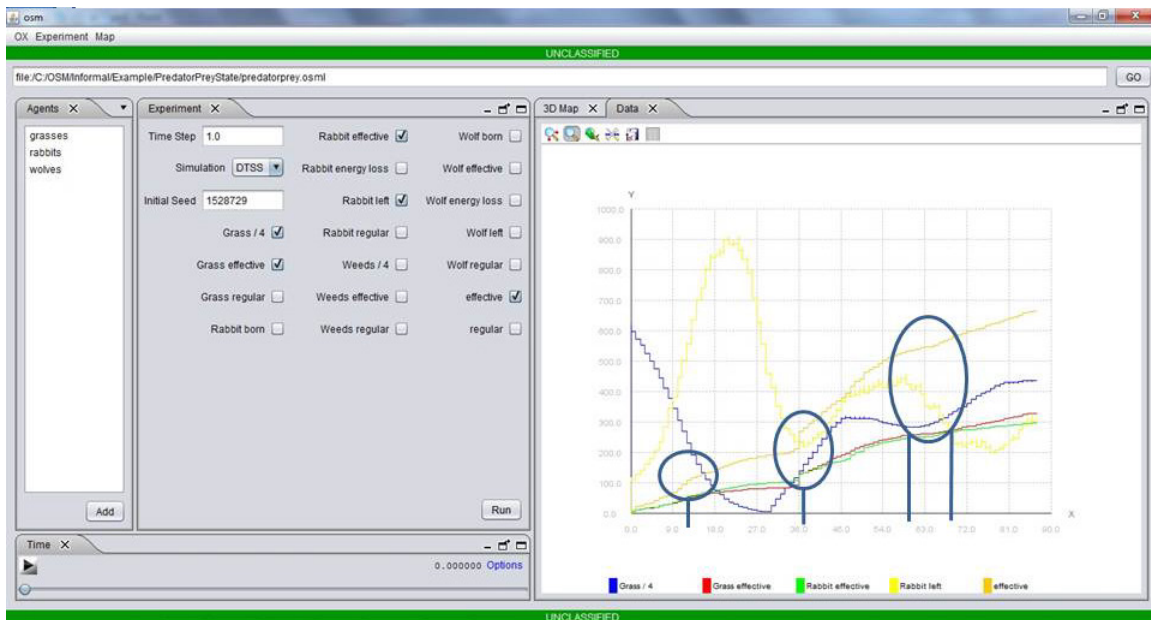
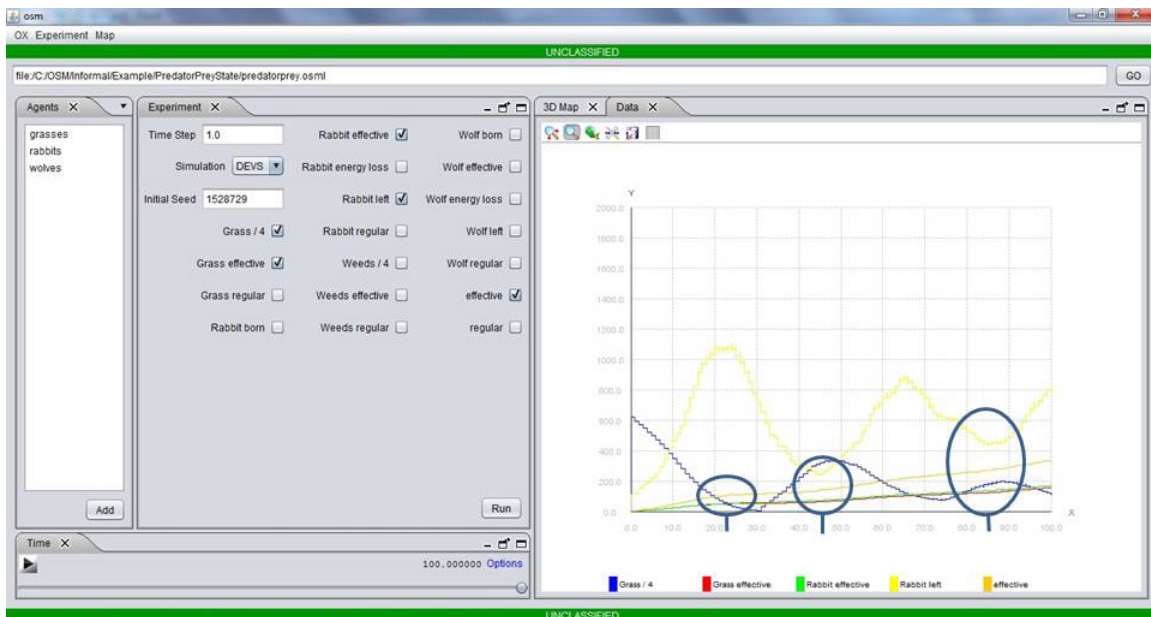


Figure 84. Graph of Metrics for DEVS Run



Figures 85, 86, and 87 show the simulation at the times circled on the DTSS graph. We found interesting groupings among the rabbits and wolves. This is known as emergent behavior. We cannot make this happen running each agent separately; and we could not account for this behavior in one agent.

Figure 85. Visualization of DTSS Run at 10 Seconds

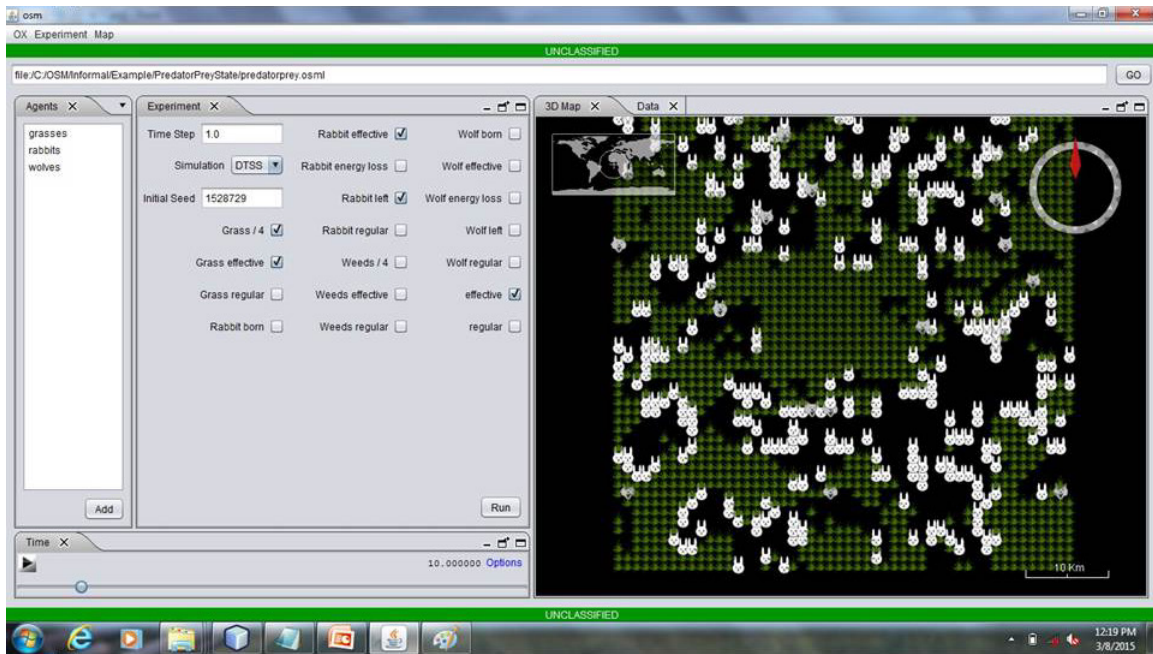


Figure 86. Visualization of DTSS Run at 37 Seconds

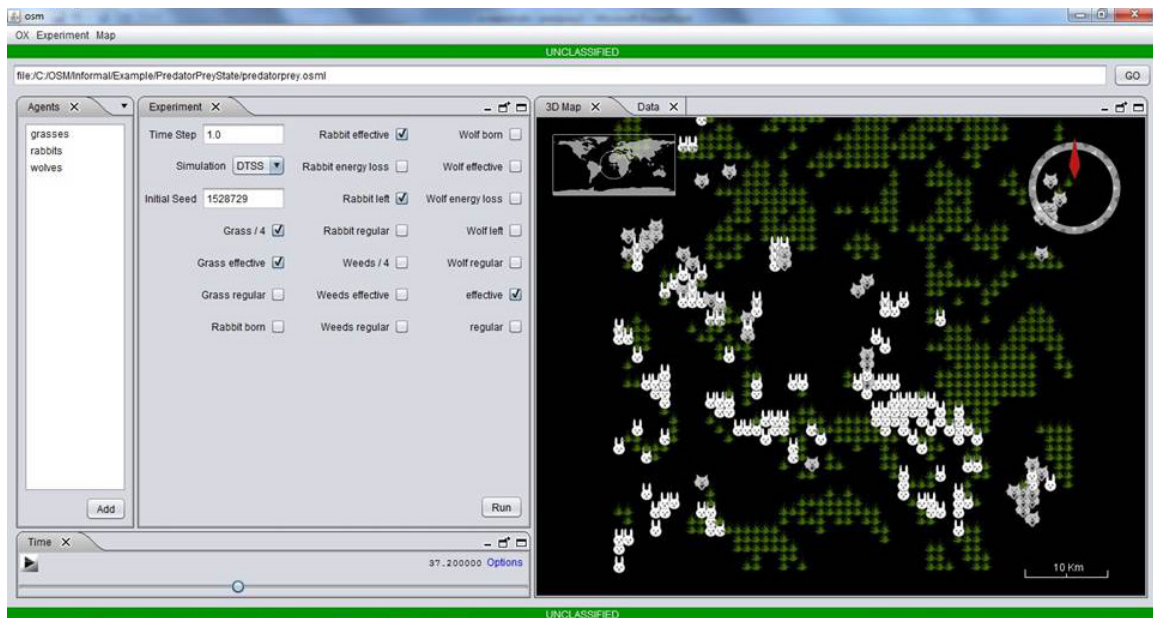
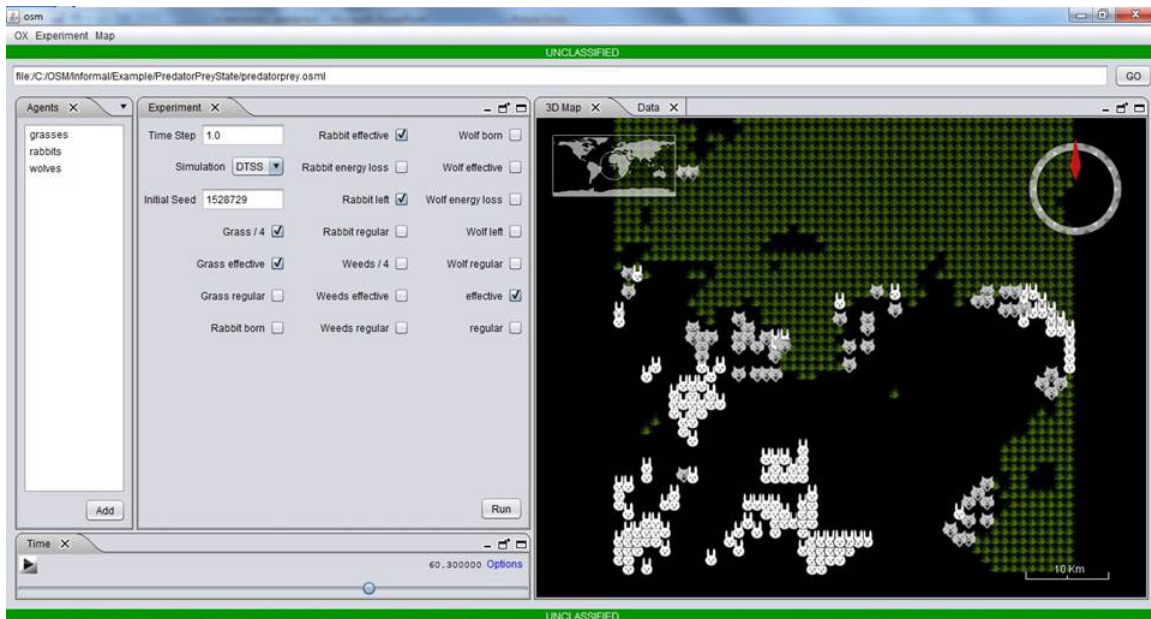


Figure 87. Visualization of DTSS Run at 60 Seconds



Figures 88, 89 and 90 show the simulation at the times circled on the DEVS graph. Because we used a different simulation method (DEVS vs DTSS), the behavior that emerged was different. This is because the behavior is driven by events vice time.

Figure 88. Visualization of DEVS Run at 22 Seconds

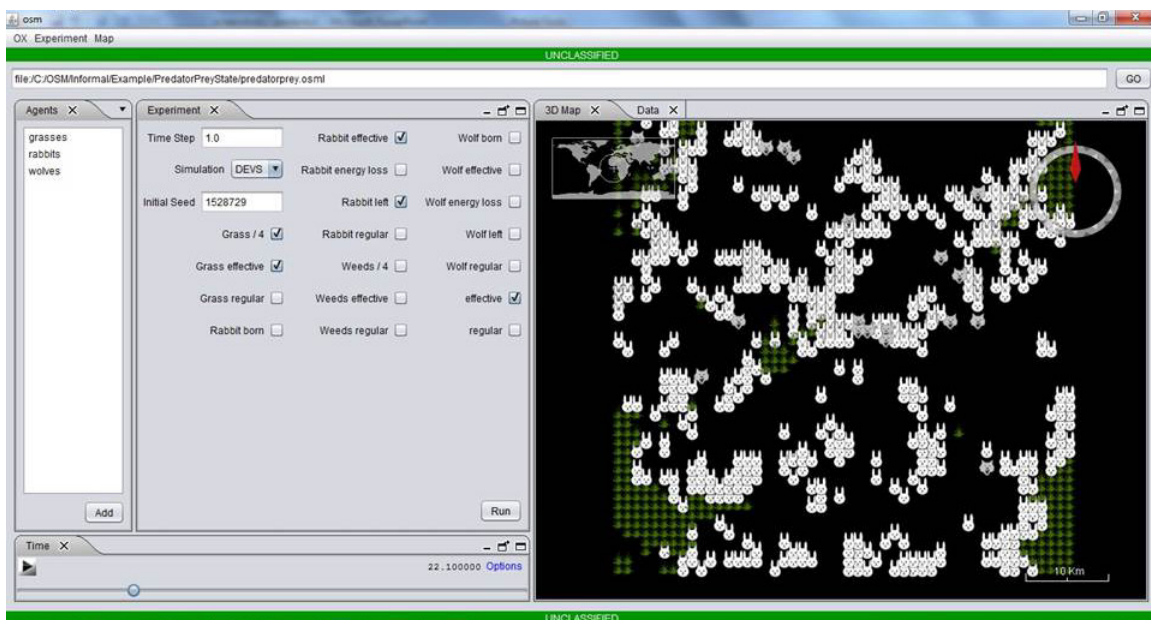


Figure 89. Visualization of DEVS Run at 45 Seconds

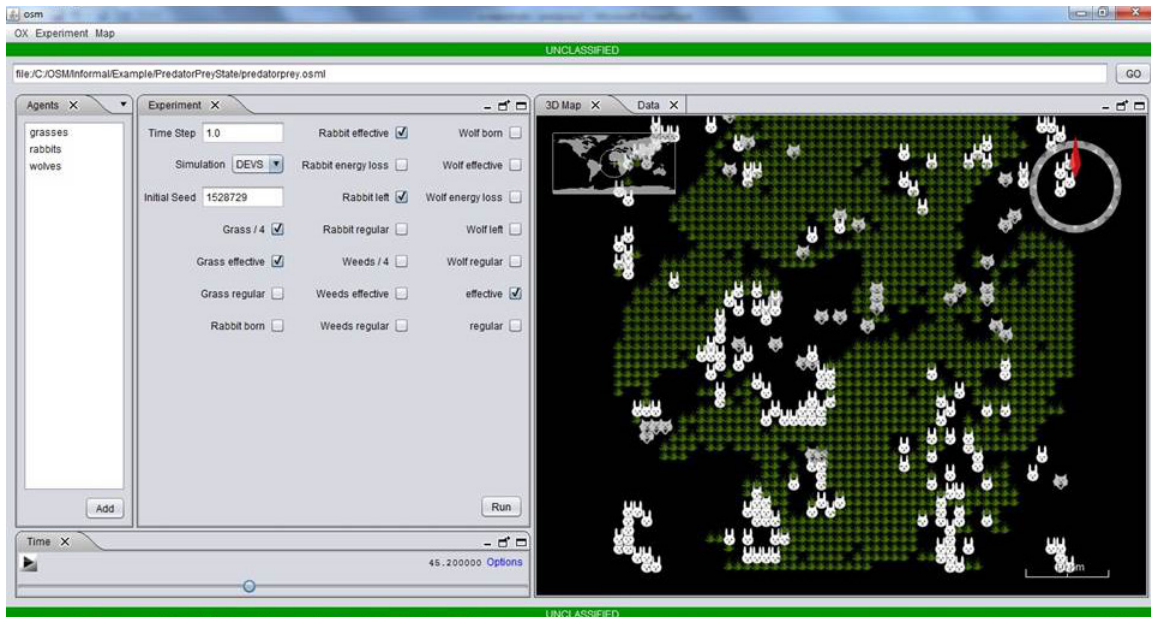
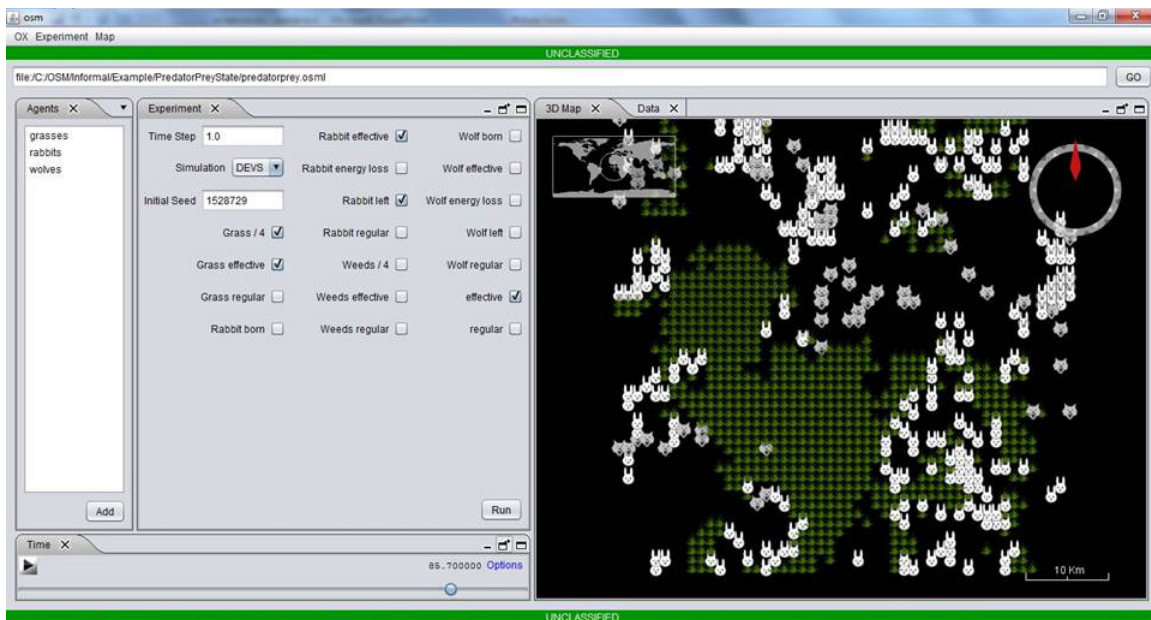


Figure 90. Visualization of DEVS Run at 85 Seconds



C. EXPERIMENTAL FRAMES

This research has created three experimental frames that can be swapped through the Experimental frame Exchange without changing the other elements of the simulation. These are: Multi-Run Experimental frame, Lattice Experimental frame, and Stochastic Experimental frame.

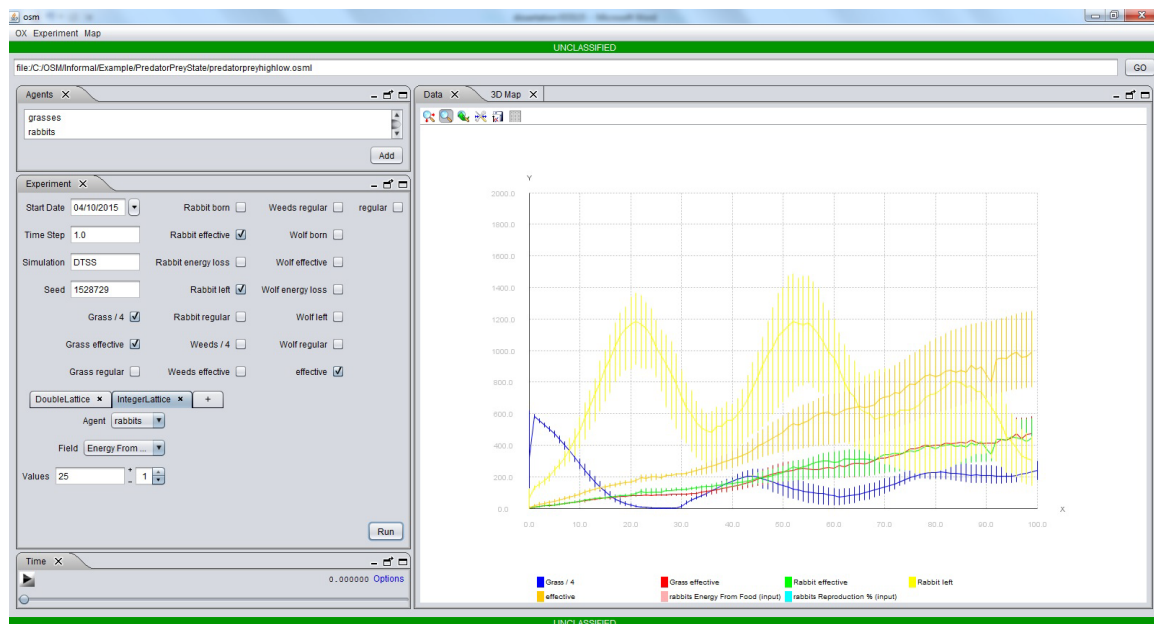
(1) Multi-Run Experimental frame

The Predator Prey simulation with closest food using the DTSS simulation type is shown in figures in previous sections using the Multi Run Experimental frame.

(2) Lattice Experimental frame

The Lattice Experimental frame allows the user to set up a set of runs using a set of numbers for chosen variables. We set up the Predator Prey simulation to make 4 runs using all combinations for rabbits to be reproduction of 18 percent and 20 percent, and starting energy units for each rabbit to be 20 and 25. The resulting graph is shown in Figure 91.

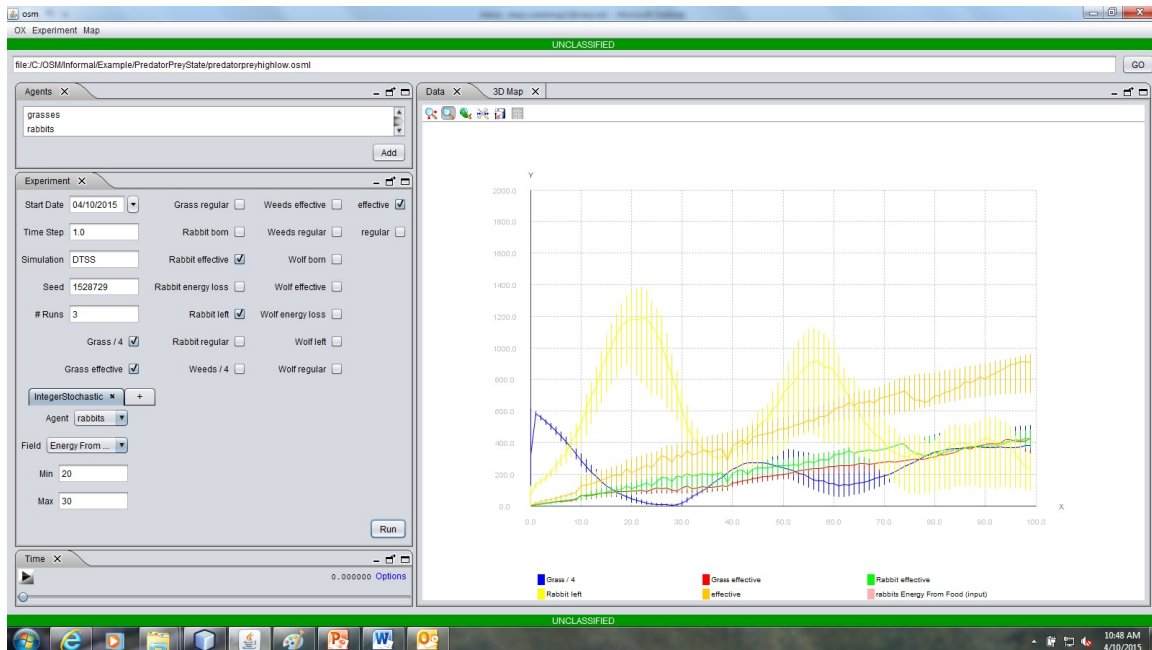
Figure 91. Graph of the Predator Prey Simulation Using a Four Run Lattice Experiment



(3) Stochastic Experimental frame

The Stochastic Experimental frame allows the user to set up some number of runs using a minimum and maximum for chosen variables. The Stochastic Experimental frame then uses a random number generator to pick random numbers for those chosen variables (within the limits the user set). We set up the Predator Prey simulation to make 3 runs setting rabbit energy units between 20 and 30 units. The resulting graph is shown in Figure 92.

Figure 92. Graph of the Predator Prey Simulation Using a Three Run Stochastic Experiment



D. PROOF OF HYPOTHESIS WITH PREDATOR PREY SIMULATIONS

The hypothesis for this work, as stated in Chapter I, is to develop an SoS M&S framework architecture that allows the identification and quantification of Emergent Behavior and will provide a:

1. Reduction of effort a programmer needs to implement changes to an SoS simulation, and
2. Reduction of time an analyst needs to set up an experiment for an SoS simulation.

Through this research, we have found that we were able

3. To reduce the effort a programmer needs to implement changes needed for the analyst's experiment by 1560 lines of code, and
4. To reduce the time an analyst needs to set up an experiment for an SoS simulation by 156 hours per experiment.

We found this by comparing the amount of time and effort needed to produce the three simulations (described in section B of this chapter) that used the Discrete Simulator (NetLogo Comparable, Discrete Event, and Discrete Time simulations) in the OSM framework without our components (swappable/reusable Experimental frames and Simulators, and the Metrics Collector) with the development of the same simulations using our components.

We make the following assumptions:

1. An average programmer can write 10 lines of code in an hour/80 lines of code in a day, and
2. An analyst will spend 2 hours defining to the programmer the changes that need to be made to an SoS simulation to implement his/her SoS experiment objectives.

We start with the amount of effort it took to create the components of this research and a Predator Prey simulation agent:

1. Simulator—220 Source Lines of Code (SLOC)
2. Experimental frame—980 SLOCs
3. Metrics Collector—360 SLOCs

This is now considered a one-time cost and will not have to be developed again. We make the assumption that this effort/SLOCs will have to be done each time the analyst wants to define a different experiment. Taken together, this will involve 1560 SLOCs/156 hours.

We found that this research will reduce the time needed for the analyst to define to the programmer what changes are needed in order for the analyst to set up/run his experiment by 156 hours per experiment because the programmer will not need to write new code for this experiment. Instead, the programmer can/will reuse or swap

Experimental frames and Simulators from other SoS simulations and will reuse the SoS Metrics Collector developed through this research. This equates to reducing the amount of code needed to be developed by 1560 SLOCs to produce an SoS simulation because of this swapping/reuse. Further, the analyst will not need to be trained on new features because he/she will be using a standard Graphical User Interface (GUI) to interface with this SoS simulation. This will result in saving another hour of the analyst' time in experiment set up.

E. BMDS SIMULATION

To show that the Simulators and Experimental frames used in the Predator Prey simulation can be reused in other simulations, a second simulation was developed. This one is of the communications between all the different elements in a Ballistic Missile Defense theater raid scenario. In this simulation, there are the following agents:

- Threat missiles firing at a defended area
- Aegis Ships (each instance a separate agent)
- Weapon Control System (WCS) on each ship
- Vertical Launch System (VLS) for each missile on each ship
- Sensor (SPY1) on each ship
- Sensor (TPY2) looking for the threat missile during flight
- Sensor (SBIRS) search for threat missile right after launch
- C2BMC controlling all weapon systems

This simulation used the discrete Simulator. We used the multi run Experimental frame. This proves the reuse claim of this research. Figure 93 shows the run.

Figure 93. BMDS Communications Simulation using the Discrete Simulator and the Multi Run Experimental frame

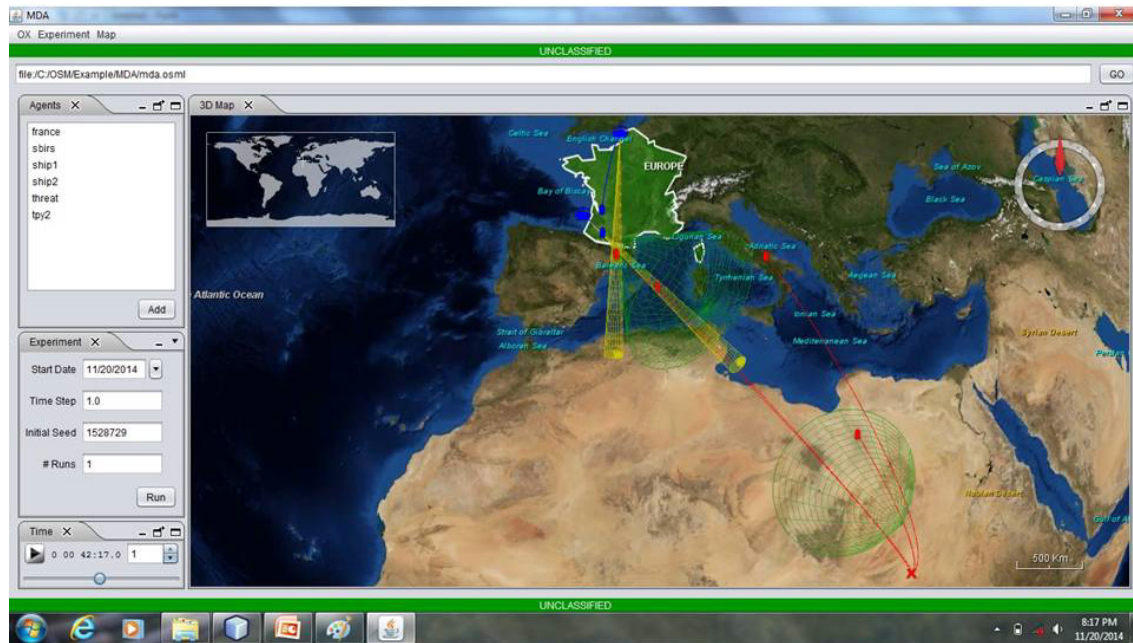
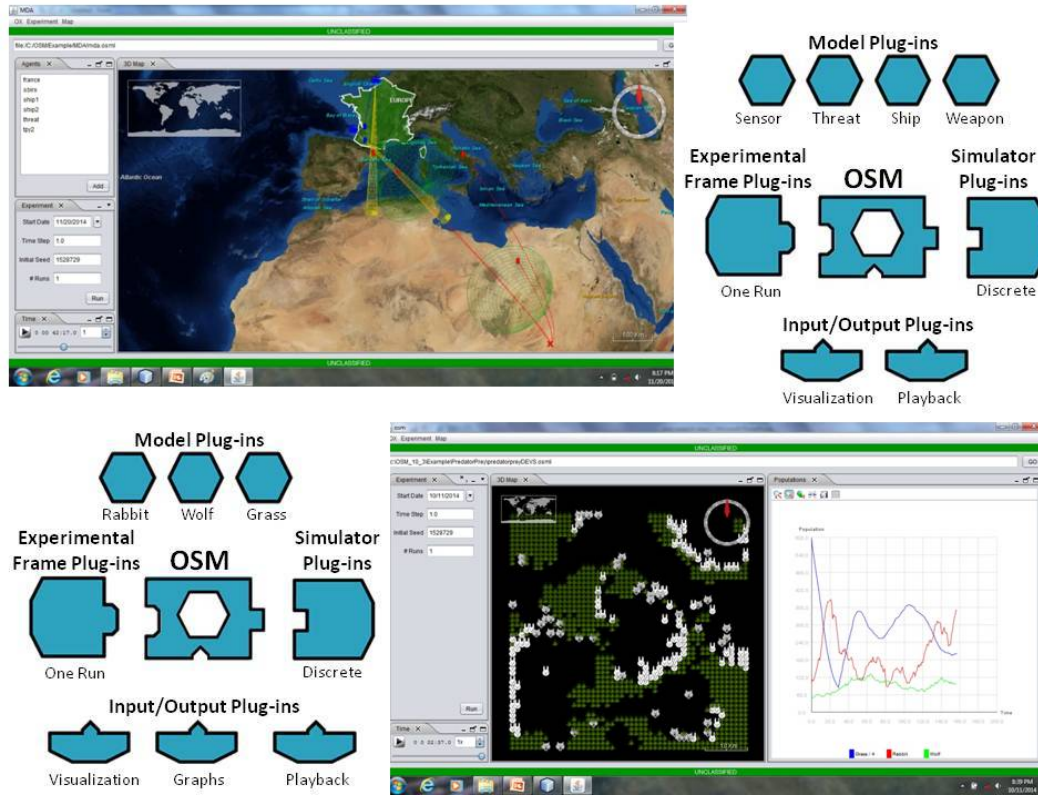


Figure 94 shows an architectural depiction of the two simulations (BMDS communications and Predator Prey). This figure shows the architectural elements used in each.

Figure 94. Architectural Elements Used in Both Simulations



F. EVALUATION

In this chapter, we have described the results of five versions of the Predator Prey simulation in order to show how the software architecture described in Chapter III can be applied. The first version was developed to baseline our Predator Prey simulation against the Predator Prey simulation example that is a part of the NetLogo product. The second and third versions were developed to show that one Simulator (known as the Discrete Simulator) can be used to drive both discrete events and discrete time stimuli. For these two versions, the agents had to be changed to pass time or events to the Discrete Simulator. The fourth and fifth versions show that the Simulator Exchange can be used to connect to either a DEVS Simulator or a DTSS Simulator without changing the agents. We could not match the NetLogo Predator Prey simulation, and in order to compare to the NetLogo Predator Prey simulation, we had to change some of the experiment data values. We believe that this is due to not having complete insight into the NetLogo infrastructure or framework code.

The Discrete Simulator was not connected to the Simulator Exchange. This was not needed because that Simulator cannot be swapped with other Simulators without changing the agents. It did allow reusability (as shown with the BMDS simulation).

The addition of another agent (in this case, a weeds agent) showed that an agent can be added after the initial development of a simulation without any of the other elements (agents, Experimental frame, Simulator, output display plug in) changing.

This work showed that the Lattice and Stochastic Experimental frames provide analysis across the simulation runs in two ways—by exporting data into Microsoft Excel to display each metric in each run and by displaying each metric for all runs together, showing the min, max, and median of all the runs for a particular metric.

This work showed that the user is given results of the runs in such a manner that the user can visually see areas on the graph where the interaction metrics combined with the agent metrics show emergent behavior. It is left to future work (described in Chapter VI) to provide a method of allowing more automated analysis of this emergent behavior.

The BMDS simulation was developed to show that the Simulators and Experimental frames used in the Predator Prey simulation can be reused in other simulations. This simulation did not use the Metrics Collector because that was not the focus.

G. SUMMARY

Contribution One, listed in Chapter I, is for a new software architecture for a SoS M&S framework that enables efficient identification and quantification of emergent behavior in an SoS M&S. We proved that we met this contribution for this work by developing 5 versions of the Predator Prey simulation and a BMDS simulation. This proved the use of swappable/reusable Experimental frames and Simulators that can be used in one SoS simulation and across SoS simulations without changing the other elements in the SoS simulation. Through the Predator Prey simulation, we also proved that we can specify and collect SoS metrics that enables the identification and analysis of

emergent behavior in the SoS simulation. This work also proved that the specifying, collecting and processing of SoS metrics is separate from the rest of the simulation code.

Contribution Two is for an improved process for an analyst to identify and quantify emergent behavior. Through the setup of the experiments for the Predator Prey simulations, this work has proven that the process steps described in Chapter IV can be done systematically and efficiently. This is possible because this work separates the following four tasks:

1. The specification, identification, and quantification of emergent behaviors via M&S metrics,
2. The specification, selection and instantiation of Experimental frames,
3. The specification, selection and instantiation of Simulators, and
4. The specification, selection, and instantiation of agents.

THIS PAGE INTENTIONALLY LEFT BLANK

VI. CONCLUSION

A. SUMMARY OF SIGNIFICANT FINDINGS

Modeling and Simulation (M&S) can be an affordable validation alternative to operationally testing all functionality and interfaces in a weapon SoS. The use of M&S can be thought of as the virtual test for SoSs, thus reducing the need for operationally testing these weapons in all possible configurations. With changes in a few parameters, M&S can be used to test many configurations (many of which are cost prohibitive to test via operationally tests). This virtual testing is not a replacement for operationally tests, but it does reduce the number of operationally tests needed. Emergent behavior brings uncertainty to this validation because this behavior occurs as a result of interactions of its components (or parts), and is not found in any of the components alone.

Many government organizations that develop weapon SoSs have a large set of M&S across various agencies that own the component weapon systems. These M&S systems represent the different component weapon systems, environments, threats, communications, debris, etc. They tie these pieces together with a framework built for that one particular simulation. The way they were built does not allow for collecting SoS metrics, that are necessary for the identification and quantification of emergent behavior of the target SoS.

Zeigler's work has principles that overcome some of the issues with SoS simulations that are in the DOD today. His work describes the basic objects of the M&S theoretical framework as the source (or real) system, the model, the Simulator and the Experimental frame. But, his theoretical formalisms do not allow for a systematic approach to collecting SoS metrics for identifying and quantifying emergent behavior in an SoS. Our research architected three components that allow for this identification, and quantification. These 3 components are the Simulator Exchange, the Experimental frame Exchange, and the Metrics Collector. These components are reusable (without change) for any SoS simulation.

The Simulator Exchange allows one Simulator (as called out in Zeigler's work) per simulation method (DEVS, DTSS, DESS, or others yet to be determined) to be used by all SoS M&S in order to provide maximum reusability for all SoS simulations, without causing change to the other elements of the M&S. We architected an interface to these Simulator such that Simulators are swappable. This means that a user can choose a simulation type at runtime without changing the other elements of the simulation.

We architected this same exchange method with the Experimental frames; we designed an Experimental frame Exchange that allows the reuse of an Experimental frame to be used by all SoS M&S. We architected an interface to these Experimental frames to be swappable. This allows any objective to be chosen at runtime for an SoS M&S without changing any of the other elements in the simulation.

These exchanges allow for the collection of SoS metrics in a Metrics Collector that interfaces with the Simulator and Experimental frame in such a way that enables the identification and analysis of emergent behavior among the interactions of the models (component systems). We believe this allows for the identification and quantification of this emergent behavior in a SoS M&S built with these components. For the purpose of this research, we defined emergent behavior as behavior of one component that is affected by another component. The Metrics Collector drives the tasks of specification and collection of the metrics through the tasks of experimental data generation (by the Experimental frame) and identification of changes to the SoS Metrics and interactions among the agents (by the Simulator) and further enable the reuse of the Experimental frames and the Simulators.

In this work, we gather SoS metrics in two ways. First, we allow each model (agent) to identify the data it wants to collect (through the Experimental frame) and alert the Metrics Collector when the values have changed using the publish/subscribe architectural style. Second, we begin with the method described in (Chan W. K., 2011) for identifying emergent behavior, which increases a counter containing the number of interactions among agents whenever an agent has to interact with another agent based on some action that the agent has to perform. Chan then graphs these interaction metrics for each run and asserts that the interaction metric deviates from a normal curve (for a

particular run) when emergent behaviors arise. For our work, we look at these interactions as one agent causing an event in another agent (such as one agent causing the destruction of another). We found that it is not just in the deviation but emergent behavior can be found in any run. We graph this interaction data in any run by overlaying the agent metrics on a graph. These interactions are identified in the Simulator (using the event driver), and analyzed and graphed (together with the SoS metrics defined in the model) in the Metrics Collector. These two sets of data, the agent metrics and the interaction metrics between the agents, are graphed together to show the relationships among the data and among the agents. This central collection point allows us to collect data across the System of Systems simulation. We have shown in this work that we can identify and quantify emergent behavior through these metrics.

We have developed a software architecture that enables the identification and quantification of emergent behavior in an SoS simulation. This architecture uses two architectural styles:

- Publish/subscribe to be used for the metrics data when set/changed and for those elements that want to know that it has changed
- Component and connector to be used for the connection between the Simulator Exchange and Simulators, and between the Experimental frame Exchange and the Experimental frames

As part of this research, we defined the process for the identification and quantification of emergent behavior in an SoS simulation that uses the software architecture defined in this work.

The Discrete Simulator was not connected to the Simulator Exchange. This was not needed because that Simulator cannot be swapped with other Simulators without changing the agents. It did allow reusability (as shown with the BMDS simulation).

The addition of another agent (in this case, a weeds agent) showed that an agent can be added after the initial development of a simulation without any of the other elements (agents, Experimental frame, Simulator, output display plug in) changing.

This work showed that the Lattice and Stochastic Experimental frames provide analysis across the simulation runs in two ways—by exporting data into Microsoft Excel

to display each metric in each run and by displaying each metric for all runs together, showing the min, max, and median of all the runs for a particular metric.

This work showed that the user is given results of the runs in such a manner that the user can visually see areas on the graph where the interaction metrics combined with the agent metrics show emergent behavior. It is left to future work (described in Chapter VI) to provide a method of allowing more automated analysis of this emergent behavior.

In summary we have met the following goals in this research:

- Development of an architectural design of a Simulator Exchange that allows the swapping of Simulators within one SoS M&S or across multiple SoS M&S without changing any of the other elements of the SoS simulation
- Development of an architectural design of an Experimental frame Exchange that allows the swapping of Experimental frames that are swappable within one SoS M&S or across multiple SoS M&S without changing any of the other elements of the SoS simulation
- Because of the reusability of the Simulator and Experimental frame Exchanges, we were able to develop an architectural design of a Metrics Collector that allows the collection of SoS metrics that can be used to identify and quantify emergent behavior in an SoS simulation
- Demonstration of this ability to identify and quantify emergent behavior via the data collected by the SoS Metric Collector.

We believe we provided the following unique contributions through this work:

- A new software architecture for a SoS M&S framework that enables efficient identification and quantification of emergent behavior in an SoS M&S. This architecture allows swappable/reusable Experimental frames and Simulators that can be used in one SoS simulation and across SoS simulations without changing the other elements in the SoS simulation. It also allows the specification and collection of SoS metrics that enables the identification and analysis of emergent behavior in the SoS simulation. This architecture separates the specifying, collecting and processing of SoS metrics from the simulation code.
- An improved process for developing SoS M&S by separating four concerns of this type of simulation: (1) the specification, identification and quantification of emergent behaviors via M&S metrics, (2) the specification, selection and instantiation of Experimental frames, (3) the

specification, selection and instantiation of Simulators, and (4) the specification, selection, and instantiation of agents.

We showed that we provided these unique contributions by through development of a Predator Prey simulation described in the NetLogo environment (Wilensky, NetLogo Wolf Sheep Predation model, n.d.) as our starting point. We developed four other versions of this Predator Prey simulation using three different Simulators and three different Experimental frames to show the swappability and reusability of these components. We further showed the reusability of the Simulators and Experimental frames by producing another simulation in a different domain using the same Simulators and Experimental frames. We did this by developing a simulation that simulates the communications between weapon systems and sensors in a BMDS theater warfare scenario.

We described the many works used as a basis for this research. This research used as its basis work done in evolutionary SoS architectures, M&S theory (specifically Zeigler's work), and emergent behavior. We surveyed many frameworks to choose the best one for this work. OSM was chosen as our framework because it meets Zeigler's M&S theory and has an evolutionary software architecture.

B. FUTURE RESEARCH

This dissertation has shown unique contributions to the field of Software Engineering. The completion of this work now allows us to determine future work that can be accomplished with this research as the basis.

This work used models on one processor, running serially. What would be the effect of using distributed, parallel models, running in separate threads?

The metrics work begun here can be greatly built upon. What causes changes in the metrics collected? We believe it is possible to determine what event and what model (agent) caused the changed in the metric. If it is an event caused by another agent, then this change would be emergent behavior as defined in Chapter II.

This work was able to prove the development of two types of swappable Simulators: DEVS and DTSS. In order to create a swappable discrete event Simulator

(known as DESS – Differential Equation System Specification), the C2 architectural style will have to be extended to go between discrete time ticks to continuous time.

This work proved that Experimental frames, using techniques for setting input parameters specifically, stochastically and in a lattice format, can be swappable. With this work as a basis, design of experiments can be introduced through the swappable Experimental frames. Design of experiments is a systematic method for determining the relationship between the inputs and the outputs. It can be used to find cause and effect relationships (Sundarajan, n.d.).

In the Modeling and Simulation community, design of experiments provides a method to determine the relationship between the inputs and the output of that simulation. It can also be thought of as being used to find cause-and-effect relationships. This can be used to find inputs that optimize the output. Another future effort building upon this research would be to make a swappable design of experiments “plug-in.”

This work uses standard interfaces and interface layers to allow agents to interact without changing other agents when an agent is added, removed, or changed. It did it by using good practices. This work can be extended to create a repository of interfaces and behaviors for a particular domain (such as Naval agent, Industry, agents, etc.)

APPENDIX. PREDATOR PREY SIMULATION PSEUDO CODE

For wolves and rabbits

```
Rabbit() {
    /* this is used in Experimental Frame */
    /* as part of setting up inputs */
    If (first rabbit (or first wolf)) {
        expFrame-> SetExpData( all exp data and their initial values);
        expFrame->SetAgentMsrdData(all msrd data);
        AddToGroup(this, "Rabbits");
    }

End

Initiate() {
    state = "hungryState";
    initial state = true;
    StateHandler();
End

StateHandler() {
    Switch(state) {
        Case "hungryState":
            /* Simulation inputs are energy start and reproduction probability */

            If (initial state) {
                Simulator->ExpDataChgd(all exp data);
                Simulator->SelectedAgentMsrdData(chosen agent measured data);
                PublishData(chosen agent measured data);

                energy = energy start;
                Initial state = false;

                /* Set variables */
                Food = closest(this, prey)

                Timestep = <from input>

            /* loop in the state without changing states*/
            simulator->ScheduleEvent( this, "moveToward," time to food);
            simulator->ScheduleEvent( this, "reproducing," time to food – energy left);
```

```

    /* Every time we enter the hungry state, we determine which states we move to -
    either eating or
    destroying*/
    simulator->ScheduleEvent(this, "targetState," timeToFood);
    simulator->ScheduleEvent("this, targetState," timeToFood-energyLeft );
Break;
Case "eatingState":
    /* Finished eating - transition to hungry*/
    simulator->ScheduleEvent(this, "targetState," 1 sec);
    simulator->ScheduleEvent(closestFoodAgent, "destroy");
Break;
Case "destroyingState":
    RemoveFromGroup(this, "Rabbits");
    Remove this agent
    --number of rabbits;
Break;
End

StateTransitionHandler() {
    Switch(state) {
    Case ("hungryState"):
        /* Entered this handler for this state at time to food seconds */
        If (this location = closest food location and time to food < energy left) {
            State = "eatingState";
        }
        Elseif (time to food > energy left) {
            State = "destroyingState";
        }
        Endif;
    Break;
    Case ("eatingState"):
        State = "hungryState";
    Break;
    }
    StateHandler();
End

EventHandler( char* event) {
    Switch(event) {
    Case ("moveToward"):
        MoveToward(closest food agent);
    Break;
    Case("reproduce"):
        /* call repeatedly */
        Reproducing(reproduction probability);
    Case("destroy"):
        state = "destroyingState";
        StateHandler();
    }
}

```

```
    Break;
    Case("targetState"):
        StateTransitionHandler();
End

Reproducing(Double reproduction probability) {
    Test for possibility of reproduction using reproduction probability
    If (reproducing) {
        Create new rabbit or wolf
        ++ no of rabbits (or wolves) born
    }
End
```

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

- Bajrachaya, K., & Duboz, R. (2013). Comparison of three agent-based platforms on the basis of a simple epidemiological model (WIP). *Proceedings of the Symposium on Theory of Modeling & Simulation-DEVS integrative M&S Symposium*. 45 #4, pp. pp. 7–12. Society for Computer Simulation International.
- Blitz, D. (1992). *Emergent Evolution: Qualitative novelty and the levels of reality*. Dordrecht: Kluwer Academic Publishers.
- Boschetti, F., Prokopenko, M., Macreadie, I., & Grisogono, A.-M. (2005). Defining and detecting emergence in complex networks. *Knowledge-based intelligent information and engineering systems*, pp 573–580.
- Buck, J. T., Ha, S., Lee, E. A., & Messerschmitt, D. G. (1994). *Ptolemy: A framework for simulating and prototyping heterogeneous systems*.
- Carnahan, J. C., Reynolds, P. F., & Brogan, D. C. (2005). Simulation-specific characteristics and software reuse. *Proceedings of the Winter Simulation Conference '05* (pp. pp. 2492–2499). IEEE.
- Chan, W. K. (2011). Interaction metric of emergent behaviors in agent-based simulation. *Proceedings of the 2011 Winter Simulation Conference* (pp. pp. 357–368). Winter Simulation Conference.
- Corning, P. A. (2002). The re-emergence of “emergence”: A venerable concept in search of a theory. *Complexity*, pp 18–30.
- Dingel, J., Garlan, D., & Damon, C. (2002). Bridging the HLA: problems and solution. *Proceedings of the Sixth IEEE International Workshop on Distributed Simulation and Real-Time Applications* (pp. pp. 33–42). IEEE.
- Director, Systems and Software Engineering and Deputy Under Secretary of Defense (Acquisition and Technology). (2008). *Systems engineering guide for systems of systems* (Vol. 1.0). Washington, DC: OSD(A&T)SSE.
- Eker, J., Jannert, J. W., Lee, E. A., Liu, J., Liu, X., Ludvig, J., et al. (2003). Taming heterogeneity-the Ptolemy approach. *Proceedings of the IEEE 91*, (pp. pp. 124–144).
- Garrett, R. K., Anderson, S., Baron, N. T., & Moreland, J. D. (2011). Managing the interstitials, a system of systems framework suited for the Ballistic Missile Defense System. *Systems Engineering*, 14, no. 1, 87–109.

- Gore, R., & Reynolds, Jr., P. F. (2008). Applying causal inference to understand emergent behavior. *Proceedings of the 40th conference on Winter Simulation* (pp. pp. 712–721). Winter Simulation Conference.
- Gore, R., Reynolds Jr., P. F., Tang, L., & Brogan, D. C. (2007). Explanation exploration: exploring emergent behavior. *Proceedings of the 21st International Workshop on Principles of Advanced and Distributed Simulation* (pp. pp. 113–122). IEEE Computer Society.
- Kim, J.-H., & Kim, T. G. (2005). Proposal of high level architecture extension. In *Artificial Intelligence and Simulation* (pp. 128–137). Springer Berlin Heidelberg.
- Kornhauser, D., Rand, W., & Wilensky, U. (2007). Visualization tools for agent-based modeling in NetLogo. *Proceedings of Agent2007*, (pp. pp. 15–17). Chicago.
- Li, Z., Sim, C. H., & Low, M. (2006). A survey of emergent behavior and its impacts in agent-based systems. *2006 IEEE International Conference on Industrial Informatics* (pp. pp. 1295–1300). IEEE.
- Macal, C. M., & North, M. J. (2008). Agent-based modeling and simulation: ABMS examples. *Proceedings of the 40th Conference on Winter Simulation* (pp. pp. 101–112). Winter Simulation Conference.
- MATLAB the language of technical computing*. (2015). Retrieved July 25, 2015, from MathWorks: http://www.mathworks.com/products/matlab/?s_tid=hp_fp_ml
- Parunak, H. V., Savit, R., & Riolo, R. L. (1998). Agent-based modeling vs. equation-based modeling: a case study and users' guide. In *Multi-Agent Systems and Agent-Based Simulation*. Springer Berlin Heidelberg.
- Privosnik, M., Marolt, M., Kavcic, A., & Divja, S. (2002). Evolutionary Construction of Emergent Properties in Multi-Agent Systems. *Proceedings of the 11th IEEE Mediterranean Electrotechnical Conference* (pp. pp. 327–330). IEEE.
- Samaey, G., Holvoet, T., & De Wolf, T. (2008). Using equation-free macroscopic analysis for studying self-organising emergent solutions. *Second IEEE International Conference on Self-Adaptive and Self-Organizing Systems* (pp. pp. 425–434). IEEE.
- Sanfelice, R. G., Copp, D. A., & Nanez, P. (2013). A toolbox for simulating hybrid systems in MATLAB/Simulink: hybrid equations (HyEQ) toolbox. *Proceedings of the 16th International conference on Hybrid Systems Computation and Control* (pp. pp. 101–106). ACM.
- Schaeffer, M. D. (2006). *Acquisition modeling and simulation master plan*. Acquisition Technology and Logistics/Office of the Under Secretary of Defense.

- Selberg, S., & Austin, M. (2008). Toward an evolutionary system of systems architecture. *Proceedings of Eighteenth Annual International Symposium of The International Council on Systems Engineering (INCOSE)*. Utrecht, The Netherlands.
- Spiegel, M. P. (2005). A case study of model context for simulation composability and reusability. *Proceedings of the Winter Simulation Conference '05* (pp. pp. 8–17). IEEE.
- Sundarajan, K. (n.d.). *Design of experiments - a primer*. Retrieved July 25, 2015, from SixSigma: http://www.isixsigma.com/tools-templates/design-of-experiments-doe/design-experiments-%E2%
- SysML Open Source Specification Project*. (n.d.). Retrieved July 25, 2015, from SysML.org: <http://sysml.org>
- Systems Tool Kit*. (n.d.). Retrieved July 25, 2015, from AGI: <http://www.agi.com/products/stk>
- Talley, D. N. (2008). *Methodology for the conceptual design of a robust and opportunistic system-of-systems*. ProQuest.
- Taylor, R. N., Medvidovic, N., & Dashofy, E. M. (2010). *Software architecture: foundations, theory, and Practice*. Hoboken: John Wiley & Sons, Inc.
- The Ballistic Missile Defense System (BMDS)*. (2015, January 15). Retrieved March 13, 2015, from U.S. Department of Defense Missile Defense Agency: <http://www.mda.mil/system/system.html>
- Tolk, A., Diallo, S., Padilla, J. J., & Turnitsa, C. D. (2011). How is M&S interoperability different from other interoperability domains? *Proceedings of the 2011 Spring Simulation Interoperability Workshop, Guest Editorial*, (p. p. 5).
- Wijesker, D., Michael, J. B., & Nerode, A. (2005). An agent-based framework for assessing missile defense doctrine and policy. *Proceedings of the Sixth IEEE International Workshop on Policies for distributed Systems and Networks*, (pp. pp. 115–118).
- Wilensky, U. (n.d.). *NetLogo*. Retrieved July 25, 2015, from NetLogo: <http://ccl.northwestern.edu/netlogo>
- Wilensky, U. (n.d.). *NetLogo Wolf Sheep Predation model*. Retrieved July 25, 2015, from NetLogo: <http://ccl.northwestern.edu/netlogo/models/WolfSheepPredation>
- Winfrey, C., Baldwin, B., Cummings, M. A., & Ghosh, P. (2014). OSM: an evolutionary system of systems framework for modeling and simulation. *Proceedings of the Spring Simulation Multi-Conference*. Tampa: Society for Computer Simulation International.

- Wooldridge, M., & Jennings, N. R. (1995). Intelligent agents: theory and practice. *The Knowledge Engineering Review*, 10(2), pp. 115–152.
- Wynne, M. W. (2004). *Policy for systems engineering in DOD*. Washington, DC: Memor by Undersecretary of Defense for Acquisition, Technology, and Logistics.
- Yilmaz, L., & Oren, T. (2004). On the need for contextualized introspective models to improve reuse and composability of defense simulations. *The Journal of Defense Modeling and Simulation: Applications, Methodology, Technology*, Vol 1 (3), pp. 141–151.
- Yilmaz, L., & Oren, T. (2009). *Agent-directed simulation and systems engineering*. Weinheim, Germany: Wiley-VCH.
- Zeigler, B. (1984). *Multifaceted modeling and discrete event simulation*. London: Academic Press.
- Zeigler, B. (2000). *Theory of modeling and simulation*, 2nd edition. San Diego: Academic Press.
- Zeigler, B. P. (1976). *Theory of modeling and simulation*. New York. John Wiley.
- Zeigler, B. P. (2013). *Guide to modeling and simulation of systems of systems*. London. Springer.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
Ft. Belvoir, Virginia
2. Dudley Knox Library
Naval Postgraduate School
Monterey, California